NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

SIGNAL CLASSIFICATION USING THE MEAN SEPARATOR NEURAL NETWORK

by

Miguel G. San Pedro

March 2000

Thesis Advisor: Monique P. Fargues
Co-Advisor: Ralph D. Hippenstiel

Approved for public release; distribution is unlimited

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** March 2000	**3. REPORT TYPE AND DATES COVERED** Master's Thesis

4. TITLE AND SUBTITLE Signal Classification Using The Mean Separator Neural Network	**5. FUNDING NUMBERS**
6. AUTHOR(S) San Pedro, Miguel G.	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)	**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**

11. SUPPLEMENTARY NOTES
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.	**12b. DISTRIBUTION CODE**

13. ABSTRACT *(Maximum 200 words)*

The explosion of digital technology provides the warrior with the potential to exploit the battlespace in ways previously unknown. Unfortunately, this godsend is a two-edge sword. Although it promises the military commander greater situational awareness, the resulting tidal wave of data impairs his decision-making capacity. More data is not needed; enhanced information and knowledge are essential.

This study built upon the Mean Separator Neural Network (MSNN) signal classification tool originally proposed by Duzenli (1998) and modified it for increased robustness. MSNN variants were developed and investigated. One modification involved input data preconditioning prior to neural network processing. A second modification incorporated projection space variance into a re-defined performance parameter and in a newly defined training termination criterion. These alternative MSNN architectures were measured against the standard MSNN, a single-layer perceptron, and a statistical classifier using data of varying input dimensionality and noise power. Classification simulations performed using these techniques measured the accuracy in categorizing data objects composed of artificial features and features extracted from synthetic communication signals. The projection space modification variant exceeded all classifiers under noise-free conditions and performed comparably to the standard MSNN in noisy environments. The preconditioned input method produced a poorer response under most situations.

14. SUBJECT TERMS Signal Classification, Neural Networks, Mean Separator Neural Network, Single Layer Perceptron, Statistical (Quadratic) Classifier	**15. NUMBER OF PAGES** 216
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UL

NSN 7540-01-280-5500

Standard Form 298 (Rev.2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

SIGNAL CLASSIFICATION USING THE MEAN SEPARATOR NEURAL NETWORK

Miguel G. San Pedro
Lieutenant Commander, United States Navy
B.S., University of California, San Diego, 1987

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
March 2000

Author: M G San Pedro LCDR, USN
Miguel G. San Pedro

Approved by:
Monique P. Fargues, Thesis Advisor

Ralph D. Hippenstiel
Ralph D. Hippenstiel, Co-Advisor

Jeffrey B. Knorr, Chairman
Department of Electrical and Computer Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The explosion of digital technology provides the warrior with the potential to exploit the battlespace in ways previously unknown. Unfortunately, this godsend is a two-edge sword. Although it promises the military commander greater situational awareness, the resulting tidal wave of data impairs his decision-making capacity. More data is not needed; enhanced information and knowledge are essential.

This study built upon the Mean Separator Neural Network (MSNN) signal classification tool originally proposed by Duzenli (1998) and modified it for increased robustness. MSNN variants were developed and investigated. One modification involved input data preconditioning prior to neural network processing. A second modification incorporated projection space variance in a re-defined performance parameter and in a newly defined training termination criterion. These alternative MSNN architectures were measured against the standard MSNN, a single-layer perceptron, and a statistical classifier using data of varying input dimensionality and noise power. Classification simulations performed using these techniques measured the accuracy in categorizing data objects composed of artificial features and features extracted from synthetic communication signals. The projection space modification variant exceeded all classifiers under noise-free conditions and performed comparably to the standard MSNN in noisy environments. The preconditioned input method produced a poorer response under most situations.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

ACKNOWLEDGEMENTS

I wish to express my gratitude

. . .to my thesis advisor, Professor Monique Fargues. This study could not have been completed without your guidance. Responsive to my six-month absence from the Naval Postgraduate School while I served on the CNO's Strategic Studies Group XVIII and the subsequent "re-think" of my thesis topic upon my return, your advice was essential to ensuring I made the necessary progress. Yet, you allowed me the freedom to chart the course of this endeavor. I am thankful for having had the opportunity to work with you.

. . .to Professor Ralph Hippenstiel for your assistance throughout this effort. It has been an honor to work with you.

. . .to my family, who always supported me. Although I do not show my appreciation nearly often enough, please know that I could not have achieved anything without you being by my side.

. . .to the memory of my wife Debbie, who showed me the true meaning of courage and strength in the face of adversity. Despite the loneliness she often tolerated as a Navy spouse and the eventual challenges she endured, Debbie always unselfishly kept my well being, success, and happiness as her primary concern. You will always be in my heart. I love you.

. . .and most importantly, to the Heavenly Father, Jesus Christ the Savior, and the Holy Spirit. Without you, Lord, nothing is possible; but with you, everything is gained.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. BACKGROUND

In "A Maritime Strategy for the Naval Century," Admiral Jay L. Johnson, Chief of Naval Operations, declared, "Just as naval forces command the operational domain of the seas, we seek to command cyberspace, by harnessing today's technology to revolutionize naval operations" (Johnson, 2000). The explosion of digital technology has indeed paved the way for the revolution in military operations currently enjoyed. Advances in undersea warfare, the cooperative engagement capability, space and terrestrial communications, and computer networks provide the warrior with the potential to exploit the battlespace in ways previously unknown. Unfortunately, this godsend is a two-edge sword. Although it gives the military commander the promise of attaining greater situational awareness, the tidal wave of data severely impairs his decision-making capacity. Instead of assisting, the data-rich, information-poor, and knowledge-starved warfighter is incapacitated and confused by the abundance of data that inundates him. More data is not needed; enhanced information and knowledge are essential.

B. THESIS OBJECTIVES

This proof of concept study continues the development of the Mean Separator Neural Network (MSNN) classification tool originally proposed by Duzenli and Fargues for identification of underwater signals, modifying it to increase performance robustness (Duzenli and Fargues, 1998). As a key component in the warfighter's observe-orient-decide-act loop, decision tools like the MSNN signal classifier promote data evolution to information. Using The MathWork's MATLAB 5, version 5.3, modification of this neural network are developed to improve its classification capabilities. The intent is to increase performance robustness and thereby improve data categorization by accounting for statistical parameters not considered in the original MSNN formulation. It is entirely expected that incorporation of these additional attributes will increase computational burden; but the effects of this extra load are expected to be unremarkable and therefore will not be rigorously monitored. The implementation of the proposed MSNN schemes

will be measured against two unrelated techniques used as benchmarks: (1) a quadratic classifier modeled purely on the statistical characteristics of the input data and (2) a single layer perceptron neural network. The accuracy of each classification method will be verified by its precision in properly typing artificial feature vectors and features extracted from simulated signal modulations. If proved successful, the altered MSNN method offers a technique that will assist the warfighter in attaining greater battlespace and infosphere acuity.

C. THESIS ORGANIZATION

Following this introduction, Chapter II presents artificial neural networks. Chapter III delves into a principal application of neural network: pattern recognition and classification. The basis of the quadratic statistical classifier, perceptron neural network, and MSNN schemes are introduced and examined. In Chapter IV, these classification techniques are tested through trial simulations. Analysis of the results provides rudimentary insight into the feasibility of each classifier. Next, Chapter V assesses the classification techniques considered by categorizing synthetic communication signals. Feature extraction is briefly discussed to emphasize this aspect of signal classification. Chapter VI summarizes the results of this study and recommends avenues for continued work.

Appendix A details an important proof of perceptron neural networks: the Fixed-Increment Convergence Theorem. Appendix B contains the empirical results of the Chapters IV and V investigations. Appendix C documents the MATLAB code written to conduct the experimental portions of this study.

II. NEURAL NETWORKS

The military commander needs advanced applications to complement the advancing appliances that have become commonplace in today's society. Indeed, Moravec claims that by the year 2030, desktop computers will have the processing power equal to the human brain (Moravec, 1999). But such capabilities are useless unless they simplify the mundane tasks dealt with on a routine basis and assist in times of crisis. For the warfighter, this amounts to creating decision aids that not only ingest data but also conveys knowledge. As a stepping stone to attaining such knowledge management capabilities, tools that communicate information to the operator, and not just delivers data, are required.

The Mean Separator Neural Network at the focus of this thesis is designed to impart information. Used as a signal classifier, this network converts raw data to useful information about the target source. But to understand how this system operates, a basic understanding of neural networks may prove useful. This chapter provides this fundamental insight into neural networks, starting with the biological inspiration for such devices: the brain.

A. BIOLOGICAL INSPIRATION

As the name implies, neural networks are structured after the workings of the brain. The question to ask then becomes why and what advantage does this provide over conventional computational devices? Indeed, studies have shown that neurons in the human brain are much slower than silicon logic gates. The computers of 1991, for example, were five to six orders of magnitude faster than the brain. Single events that take nanoseconds in computers to process, require milliseconds in the cerebral cortex. Yet, it is common knowledge that the human brain is more powerful than even today's computers. For instance, perceptual recognition takes 100-200 milliseconds for people, but requires days for computers. In accomplishing such tasks, the brain is also much more energy efficient. While computers consume 10^{-6} Joules/sec per operation, the energy expenditure of the brain is only 10^{-16} Joules/sec per operation. If computers

process individual instructions more quickly than the brain, how does the biological neural network operate more efficiently?

The brain achieves such performance levels by utilizing a highly complex, non-linear network of parallel processing units. Nearly a quadrillion (10^{15}) connections link the one hundred billion processing elements (called neurons) that make up the brain. Shown in Figure II-1, these neurons are composed of three principal components. The dendrites, the axon, and the cell body. The dendrites and axons are the communication lines that convey electro-chemical messages between adjacent neurons. Dendrites are the receptive appendages; axons, the transmission appendages. The connections formed by these components are the brain's synaptic links. Between the dendrites and axon, information is processed by the cell body. The arrangement of the neurons, the strength of the synaptic links, and the summing and thresholding of the cell body determines the processing power of the biological neural network. (Haykin, 1994, pp. 1-4), (Hagan, et al, 1996, pp. 1-8 – 1-9)

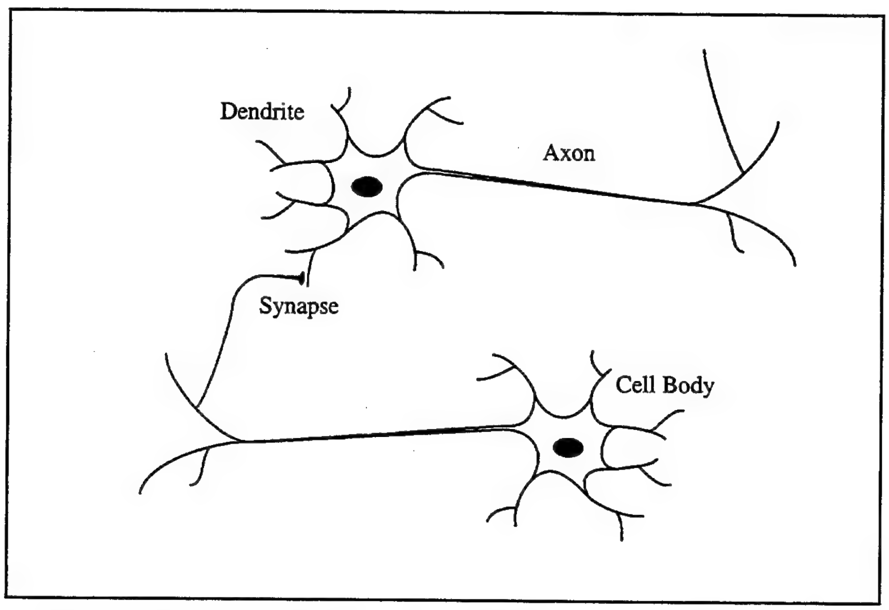


Figure II-1. Biological Neuron. From Ref. [Hagan, et al, 1996, p. 1-8]

B. COMPUTER IMITATION

Because of its massively parallel and complex structure, the brain operates more efficiently than conventional computers. It is this capability that artificial neural networks strive to replicate. Like the anatomical prototype, artificial neural networks use experiential knowledge to understand and interact with the environment. That is, artificial neural networks learn. The artificial network process input data to approximate a situation and stores this learned information as "synaptic" weights. Hence, an artificial processing element can be modeled after the biological neuron, as shown in Figure II-2. In this diagram, the weighted input link, **w**, replace the dendrites and synapses; a linear summer and a non-linear activation function, φ, the cell body; and the output link, **a**, the axon. As a result, the artificial neuron output, defined in Figure II-2 as

$$\mathbf{a} = \varphi(\mathbf{w}^T \bullet \mathbf{p} + b), \tag{2.1}$$

illustrates that the non-linear activation function, like the cell body, determines the neuron's characteristic ability to solve specific problems.

Using this basic building block, parallel-processing networks can be constructed. Feeding the same input to several neurons results in a network layer of parallel

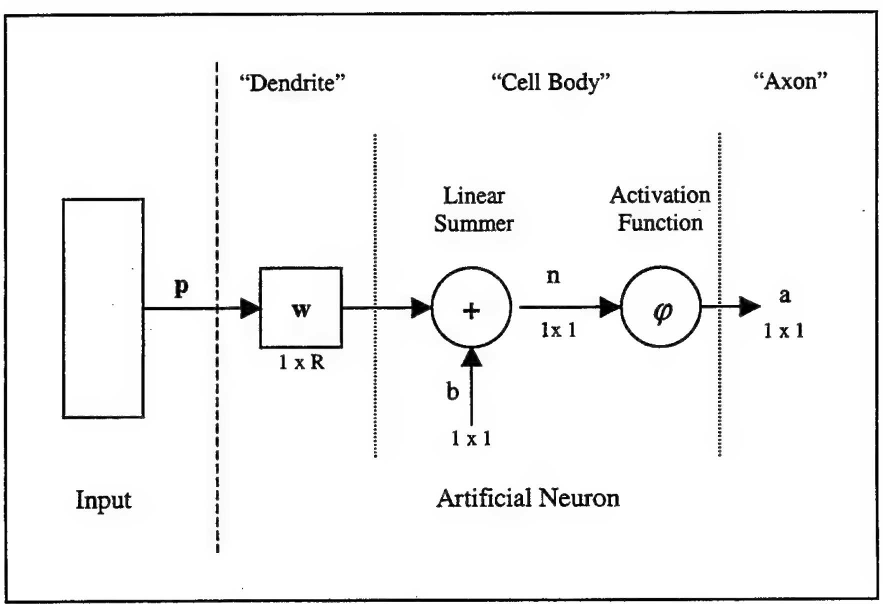


Figure II-2. Artificial Neuron. After Ref. [Hagan, et al, 1996, p. 4-4]

processing elements. The data input to these processing element could be a vector or matrix of information originating from an external sensor or an internal storage device. But, when this feed comes from an upstream neural layer, or alternatively, when the layer output supplies a subsequent downstream network layer, complex network structures are assembled. Thus, even though current neural network architectures fall short of the physiological capabilities, artificial neural networks begin to resemble the human brain.

With this model of an artificial neuron, a single-layer Mean Separator Neural Network will be built and examined. Further details on neural networks can be obtained by consulting listed references (Dayhoff, 1990), (Fausett, 1994), (Hagan, et al, 1996), (Haykin, 1994).

III. CLASSIFICATION

Chapter II briefly discussed neural network fundamentals. In Chapter III, a specific application of this computational tool will be considered.

Adept at solving problems, neural networks are being used in a growing number of diverse fields. In addition to applications in engineering, mathematics, and the physical sciences, they have proved useful in medicine, banking and finance, and literature. Table III-1 lists a few of the fields impacted by neural network advancements.

INDUSTRY	APPLICATION
Aerospace	Flight Path Simulation Aircraft Control Component Simulation and Fault Detectors
Automotive	Automatic Guidance Systems
Banking	Document Readers Credit Application Evaluations
Defense	New Sensors Target Tracking and Weapon Steering Object Discrimination
Electronics	IC Chip Layout and Process Control Failure Analysis Code Sequence Prediction
Entertainment	Animation Special Effects
Finance and Securities	Market Analysis and Forecasting Real Estate Appraisal Credit Line Use Analysis
Insurance	Policy Application Evaluation
Medical	EEG and ECG Analysis Breast Cancer Cell Analysis Hospital Quality Improvement
Oil and Gas	Exploration
Robotics	Manipulator Controllers Vision Systems
Speech	Speech Recognition and Compression Text to Speech Synthesis
Telecommunications	Image and Data Compression Real-Time Language Translator Automated Information Services

Table III-1. Neural Network Applications. After Ref. [Hagan, et al, 1996, pp. 1-5 - 1-6].

Common among these applications is a reliance on the neural network's natural ability to recognize patterns. As a result, neural networks are commonly tasked with separating data into a finite number of classes, i.e., classifying. *Classification* is the task of categorizing observation into distinct groups based on characteristics of the class. For example, when separating fruit, shape, weight, size, color, texture, or smell could be used to differentiate oranges from apples or bananas.

The attributes used to separate the distinct classes are called *features*. These features, arranged as vectors, comprise the problem's input or data space. Although it may seem that the likelihood of correct classification increases with higher feature space dimensionality, this is not necessarily the case. For instance, consider a person wishing to purchase an automobile. He may convey to a dealer in meticulous detail the specifications he desired (e.g., exterior color, type of interior, engine horsepower, gas mileage, trunk capacity, wheel base length, audio components, etc.) so as to identify a particular vehicle. Imagine the dealer's exasperation as the customer goes through this litany. The main disadvantages of the precision characterized in this example are

1. irrelevant and/or noisy features may be taken into account,
2. a requirement for a large sample to assess the robustness of the features used.

In addition, relying on such a large feature space increases the computational load and, consequently, processing time of the problem. (Duzenli and Fargues, 1998)

But alternatively, consider the overzealous salesman who bombards a customer with countless questions without receiving any satisfactory answers in return. Often, the particular pieces of information needed may not be obtainable. Solving the classification dilemma thereby becomes a problem of identifying an algorithm that will type observations to the correct class when only a reduced feature space, either by design or as dictated by the situation, is available.

Feature determination and extraction are vital aspects of the classification problem; however, the main emphasis of this thesis will be algorithm identification and testing. As will be seen, the method by which neural networks classify is dependent on the algorithm used. But, by no means are neural networks the only tool used to separate

data into proper classes. In a paper presented at the 1999 Military Communication International Symposium, Sills identifies methods studied to classify modulated signals. These efforts focused on frequency-domain parameters (Ghani and Lamontagne, 1993), (Lallo, 1999); statistical attributes of various signal parameters (Sills, 1999); and higher order statistics of cyclostationary signals (Reichert, 1992). With regards to neural networks, these parameters could constitute the features of interest.

Specifically, this thesis continues the development of the Mean Separator Neural Network (MSNN) originally proposed by Duzenli and Fargues for classifying underwater signals (Duzenli and Fargues, 1998). To gauge its performance, the MSNN classification capability was measured against a single-layer perceptron neural network – the least complex neural network used for classification – and a classifier based solely on the statistical characteristics of a particular class. This statistical classifier is considered next.

A. STATISTICAL CLASSIFIER

A *statistical classifier* served as one benchmark for the results obtained in this study. Statistical classifiers model the problem space based on data attributes (such as mean, covariance, or any higher order moment). Consequently, they may also be known as parametric classifiers. Non-parametric classifiers, on the other hand, approximate the problem based on actual empirical data. Neural networks are non-parametric classifiers.

For this study, the statistical classifier used was the *quadratic classifier* derived from the Bayes likelihood ratio, which has been shown to minimize error probability (Fukunaga, 1990, p. 124). The formulation of the decision rule governing the quadratic classification algorithm follows.

Consider a space composed of m classes, namely π_1, π_2, π_3, . . . π_m. At some time, an observation **x** belonging to class π_i occurs. The decision rule will classify **x** to π^* so as to minimize error; that is classify **x** to $\pi^* \equiv \pi_i$. Setting the loss function for this situation as

$$\lambda(\pi_i \mid \pi_j) = \begin{cases} 0 & i = j \\ 1 & i \neq j \end{cases} \tag{3.1}$$

implies that no loss arises when correct classification occurs, while unit loss results from improper classification. From Equation 3.1, the decision rule is given by

$$\pi^*(\mathbf{x}) = \pi_i \quad \text{if} \quad P(\pi_i \mid \mathbf{x}) > P(\pi_j \mid \mathbf{x}), \quad \forall j, \quad j \neq i. \tag{3.2}$$

Using Bayes' Rule to rewrite the conditional *a posteriori* probabilities in terms of the density function $p(\mathbf{x}|\pi_k)$ and the *a priori* probabilities P_k leads to

$$\pi^*(\mathbf{x}) = \pi_i \quad \text{if} \quad p(\mathbf{x} \mid \pi_i)P_i > p(\mathbf{x} \mid \pi_j)P_j, \quad \forall j, \quad j \neq i. \tag{3.3}$$

For a two class (i =1, j=2) multi-variant normal system, $p(\mathbf{x}|\pi_k)$ can be expressed as

$$p(\mathbf{x} \mid \pi_k) = \frac{1}{|2\pi\boldsymbol{\Sigma}_k|^{1/2}} \exp\left[-\tfrac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}_k^{-1}(\mathbf{x}-\boldsymbol{\mu}_k)\right] \tag{3.4}$$

with Σ the class covariance matrix, μ the class mean vector, and $\mathbf{x}$ the observation. Substituting Equation 3.4 into the inequality of Equation 3.3 yields

$$\begin{aligned} \pi^*(\mathbf{x}) = \pi_1 \quad \text{if} \quad & \frac{1}{|2\pi\boldsymbol{\Sigma}_1|^{1/2}} \exp\left[-\tfrac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_1)^T \boldsymbol{\Sigma}_1^{-1}(\mathbf{x}-\boldsymbol{\mu}_1)\right] P_1 \\ & > \frac{1}{|2\pi\boldsymbol{\Sigma}_2|^{1/2}} \exp\left[-\tfrac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_2)^T \boldsymbol{\Sigma}_2^{-1}(\mathbf{x}-\boldsymbol{\mu}_2)\right] P_2. \end{aligned} \tag{3.5}$$

Since both sides of the inequality are positive, taking the natural logarithm of each term in Equation 3.5 results in

$$\ln\frac{1}{|\boldsymbol{\Sigma}_1|} - (\mathbf{x}-\boldsymbol{\mu}_1)^T \boldsymbol{\Sigma}_1^{-1}(\mathbf{x}-\boldsymbol{\mu}_1) + 2\ln P_1 > \ln\frac{1}{|\boldsymbol{\Sigma}_2|} - (\mathbf{x}-\boldsymbol{\mu}_2)^T \boldsymbol{\Sigma}_2^{-1}(\mathbf{x}-\boldsymbol{\mu}_2) + 2\ln P_2. \tag{3.6}$$

Alternatively, Equation 3.6 can be expressed as

$$\ln|\boldsymbol{\Sigma}_2| + (\mathbf{x}-\boldsymbol{\mu}_2)^T \boldsymbol{\Sigma}_2^{-1}(\mathbf{x}-\boldsymbol{\mu}_2) - 2\ln P_2 > \ln|\boldsymbol{\Sigma}_1| + (\mathbf{x}-\boldsymbol{\mu}_1)^T \boldsymbol{\Sigma}_1^{-1}(\mathbf{x}-\boldsymbol{\mu}_1) - 2\ln P_1. \tag{3.7}$$

When Equations 3.6 or 3.7 are true, observation $\mathbf{x}$ is categorized as belonging to class π_1.

Considering the original problem of m classes, the decision criteria is stated here as Equation 3.8:

$$d_i(\mathbf{x}) = \ln|\boldsymbol{\Sigma}_i| + (\mathbf{x}-\boldsymbol{\mu}_i)^T \boldsymbol{\Sigma}_i^{-1}(\mathbf{x}-\boldsymbol{\mu}_i) - 2\ln P_i. \tag{3.8}$$

Therefore, using the mean vector and covariance matrix of each class, m decision values can be calculated for the observation **x**. The correct classification of **x** is the class that gives the lowest value for d. (Brunzell and Eriksson, 1999)

Unfortunately, Equation 3.8 requires that the data set be normally distributed. When this is the case, the quadratic classifier performs remarkably well.

B. PERCEPTRON

1. Principles of Operation

Inspired by the assertion that "in spite of its apparent simplicity, the (single layer perceptron) trained by adaptive optimization techniques is in fact a very rich family of linear classifiers," the second benchmark used to gauge the MSNN performance was a perceptron neural network (Raudys, 1996). Developed in the 1950s by Frank Rosenblatt, perceptrons are designed to linearly separate adjacent class groups (Figure III-1). Each boundary in Figure III-1 is determined using a separate perceptron component, shown in Figure III-2. In this figure, the hard limit layer represents the actual processing element. The input block, comprised of R-dimensional vectors, **p**, corresponds to the training or observation data. For R greater than two, the decision boundary shown in Figure III-1

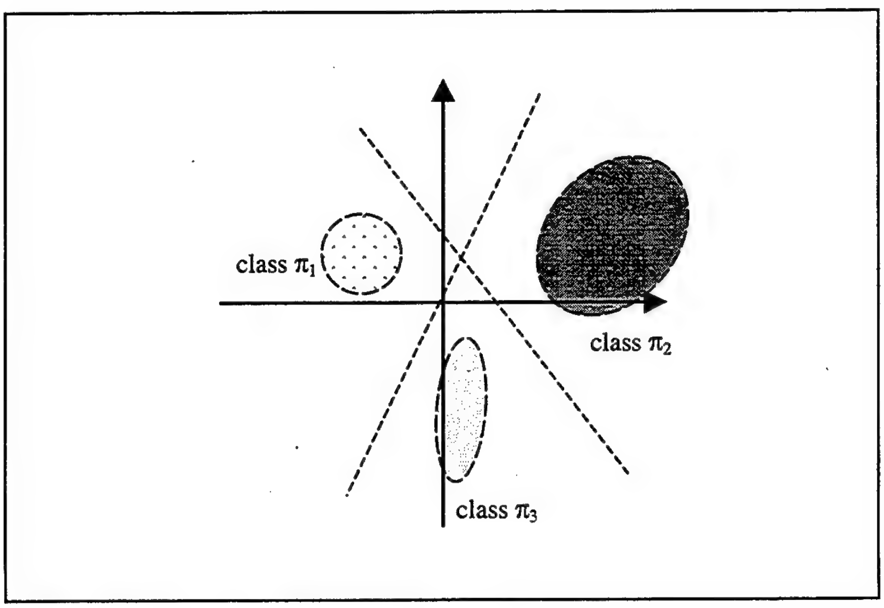


Figure III-1. Linearly Separable Classes.

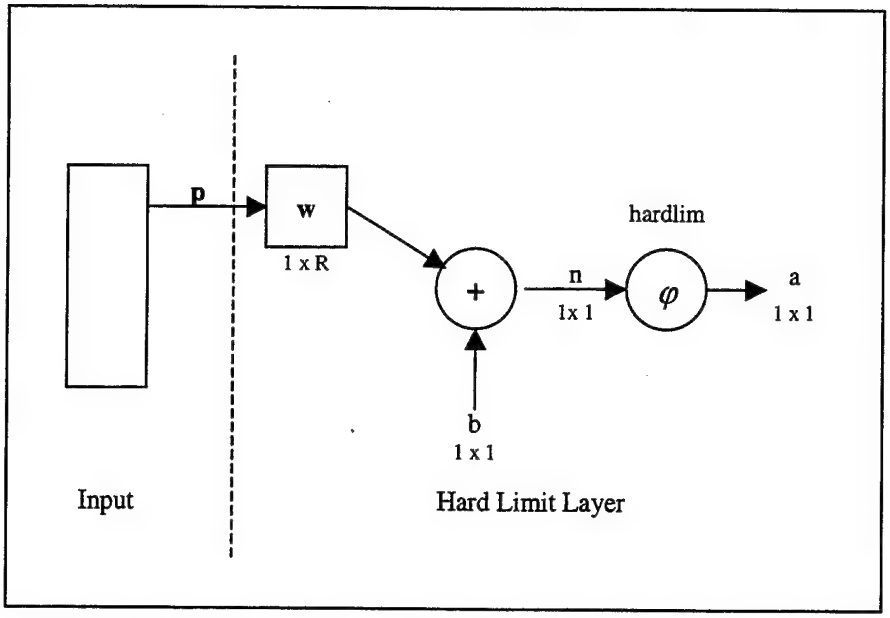


Figure III-2. Single Perceptron Processing Element. After Ref. [Hagan, et al, 1996, p. 4-4]

becomes a hyperplane. The weight row vector, **w**, and bias scalar, b, transform the input observations into a scalar output n, which is then non-linearly mapped by the activation function, φ. The perceptron output therefore equates to

$$a = \varphi(\mathbf{w} \bullet \mathbf{p} + b). \tag{3.9}$$

The activation function φ normally used for the perceptron is the hard limit, or *hardlim*. Figure III-3 illustrates the characteristic of this transformation.

As shown in Figure III-3, the only possible outputs of a single perceptron neural network are 0 and 1. Consequently, the neural network can only separate two classes; the decision boundary, for example, isolating class π_1 (network output 0) from class π_2 (network output 1).

This decision boundary is specified by the *hardlim* argument and is represented mathematically by the linear equation

$$\mathbf{w} \bullet \mathbf{p} + b = 0. \tag{3.10}$$

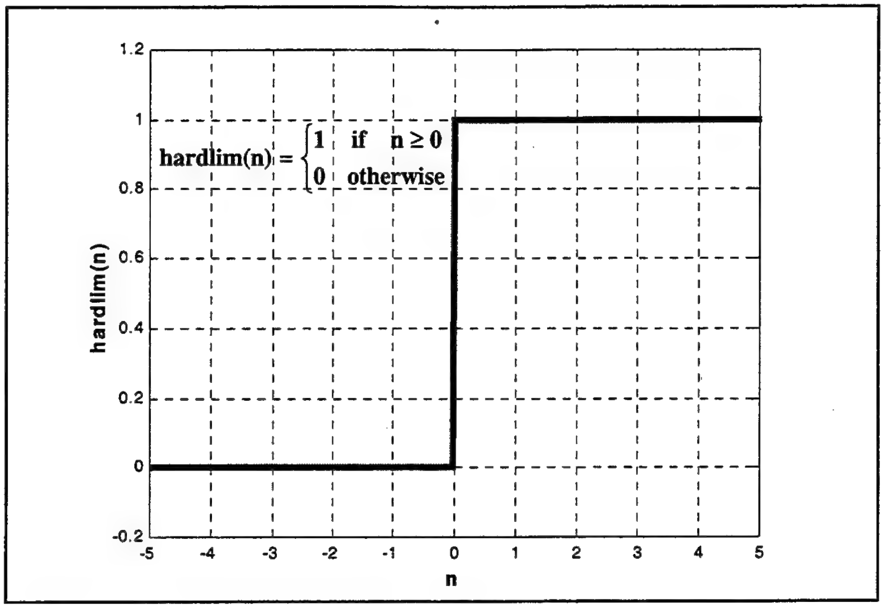


Figure III-3. ***hardlim*** **Activation Function.**

If the inner product of the input vector **p** and the weight vector **w** is greater than –b, the *hardlim* non-linear transformation will map to 1; if the inner product is less than –b, *hardlim* will map to 0. This provides the distinction needed for classifying observations.

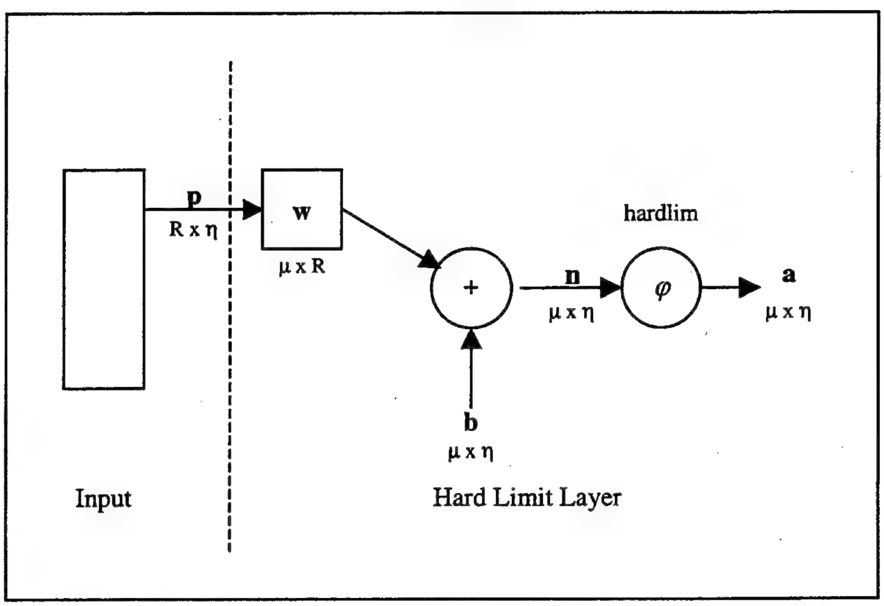


Figure III-4. Multiple Perceptron Neural Network. After Ref. [Hagan, et al, 1996, p. 4-4]

Since each perceptron can distinguish only two different classes, classification problems involving more than two choices require a multiple-neuron architecture. μ (rounded up to the next integer) perceptrons are needed to classify 2^{μ} different classes. The three-class case shown in Figure III-1, for instance, requires two processing elements. Using matrix-vector notation, Figure III-2 can be modified to illustrate the general case of a multi-perceptron architecture and multiple trials, η (Figure III-4).

With μ processing elements, the decision rule for multi-neuron networks must consider a μ-dimensional output vector of 1s and 0s. Each unique combination of 1 and 0 corresponds to a particular class. The typing of an input observation is determined by matching the neuro-classifier output to one of these different sequences. Unfortunately, when the number of possible bit strings exceeds the number of classes, the input data may type to a non-class sequence. This frequently occurred during the simulations discussed in Chapter IV and V.

In summary, as an observation is processed through a trained perceptron network, the classifier output will identify the appropriate class type for both single and multiple neuron cases. Training the neuro-classifier to determine the proper output is discussed next.

2. Training

Prior to implementing the perceptron neuro-classifier, the network must be trained to recognize different classes. This training is accomplished through a supervised learning approach in which sets of input data and corresponding target output are presented to the neural network. The network batch processes the input observations for comparison of the resulting output to the desired output. A difference error between these two output values is calculated and used to update the perceptron parameters – the network's weight vector and bias. Since the network can only output 0 or 1, the error generated is limited to either 0 or ±1 (or, for multi-perceptron networks, a vector of 0s and ±1s). If the error is zero, no weight or bias update occurs.

When the error is non-zero, the weight vector is updated by adding a correcting term (the product of the error and input data) to the weight vector. For the bias, the error

is simply added to the bias. Mathematically, Equation 3.11 and 3.12 compactly show this perceptron learning rule for the general case of multiple neuron networks as

$$\mathbf{w}^{new} = \mathbf{w}^{old} + \mathbf{e} \cdot \mathbf{p}^T = \mathbf{w}^{old} + (\mathbf{t} - \mathbf{a}) \cdot \mathbf{p}^T \tag{3.11}$$

$$\mathbf{b}^{new} = \mathbf{b}^{old} + \mathbf{e} = \mathbf{b}^{old} + (\mathbf{t} - \mathbf{a}). \tag{3.12}$$

These operations improve classification performance by adjusting the slope and position of the perceptron decision boundary towards the input data point. In doing so, the linear separator incrementally rotates and translates to place the input data on the correct side of the decision boundary.

3. Training Termination

An iterative process, perceptron training involves cycling through the input/target output pairs – each iteration through the entire data set constituting an *epoch* – until network *convergence*. Here, convergence refers to reaching and maintaining a steady state error condition. For linearly separable classes, perceptron training results in the best case, zero-error solution within a finite number of epochs (see Appendix A).

Unfortunately, linearly separable problems are an ideal classification case. Convergence, in general, does not imply a zero-error final state as the nature of the classification problem may dictate that the steady state solution includes a constant error level. Or, as another possible outcome, the neural network may not converge at all, but instead oscillate or erratically deviate about a fixed value. And finally, even when the network converges, there is no guarantee that this constant state will be attained within a reasonable time period. For these less than optimal cases, termination parameters signal when to stop network training. Typically these parameters are satisfied by reaching a maximum number of epochs or a maximum acceptable performance level.

The simplest approach to end network training would be to reach a prescribed maximum number of training cycles. When properly chosen, this epoch limit can assure attaining an adequate solution. Unfortunately when specified too low, unsatisfactory network output may result since the network would not have had sufficient time to achieve an acceptable final weight and bias. Conversely, fixing the maximum epoch

setpoint too high would increase the likelihood of adequate training but at the cost of an excessively long training period.

But, determining the number of epochs required to obtain an optimal solution hinges on specifying what is meant by "optimal" solution. To define "optimal" in this sense requires having *a priori* information of the input data distribution. For a linearly separable classification problem, an optimal solution would lead to zero-error. For other situations, a predetermined metric specifying an acceptable error limit, such as a maximum mean squared error or sum of squared errors, could be used to end network training. Regardless of the termination parameter used, prior knowledge of the input data allows better approximation of the maximum epoch limit. Combining this maximum number of iterations with an appropriately set performance measure provides for adequate control of the training length.

4. Limitations

Section III.B has dealt with using the perceptron neural network for classification purposes. Through a simple learning rule (Equations 3.11 and 3.12), perceptrons can classify to zero-error solutions in a finite amount of time. Unfortunately, as linear classifiers, perceptrons accomplish this only for linearly separable cases. As a result, perceptron networks rarely converge to zero-error solutions, thus requiring the implementation of termination parameters to limit network training.

This, however, is not the principle disadvantage of the perceptron network. Recalling that the perceptron uses the *hardlim* transform, the network's piecewise continuous, hence non-differentiable, activation function does not allow application of mathematical optimization techniques. Solving classification problems, therefore, becomes tedious as the iterative process amounts to "hunting-and-pecking" for the best fit (i.e., smallest error) solution. This trial-and-error method limits perceptron efficacy.

Yet despite these inadequacies, improvements in perceptron efficiency are possible with multiple layer network design. The next section, however, will show that by design the MSNN is a single layer neural network. Because of the focus on this architecture, this investigation only considered single layer perceptron networks.

C. MEAN SEPARATOR

As previously mentioned, classification requires (1) the extraction and reduction of features that characterize the distinct categories and (2) the application of an analytical tool that evaluates and separates observations. This thesis, concerned principally with the latter requirement, is focused on the Mean Separator Neural Network (MSNN) originally presented by Duzenli and Fargues (1998). In addition, three variations to this standard mean separator algorithm were investigated to determine if enhanced system performance and robustness could be achieved.

1. Principles of Operations

The MSNN differentiates two classes by evaluating one-dimensional projections of each data distribution onto varying axes to ascertain which transformation direction maximizes the spread between the class mean values; hence the term "mean separator." Figure III-5 illustrates this concept in two-dimensional space by showing two possible mean separator projection axes. The ellipses represent two classes and the shading within each conveys the data distribution; the darker regions being more densely populated than

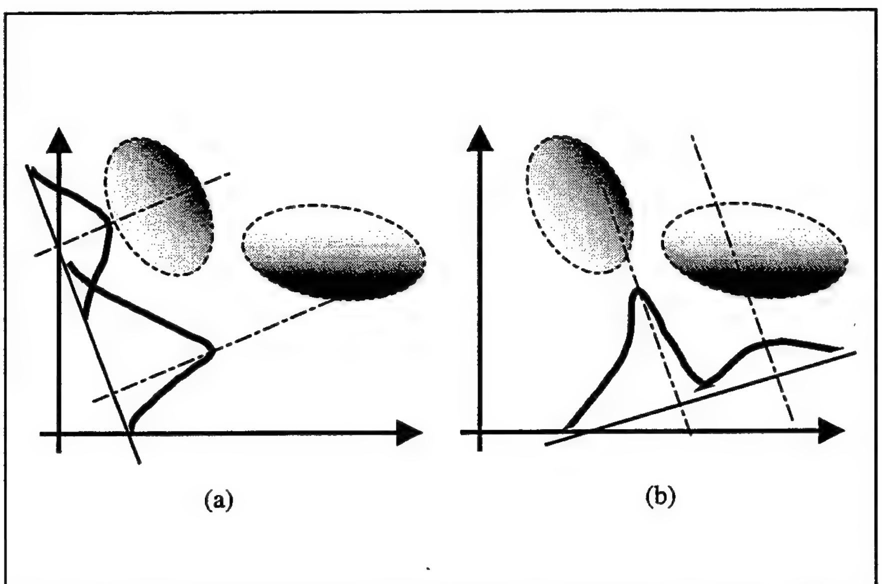


Figure III-5. MSNN Projection. Projection lines and data distribution. Due to greater mean separation, (a) is the preferred projection.

the lighter. The orthogonal axes correspond to two elements of the feature space. The slanted solid lines indicate the projection axes and the slanted dashed lines are the projection of the class means onto these axes.

Of the two projections shown in Figure III-5, case (a) with the larger mean separation depicts the preferred selection. Class typing of future observations would then entail projection of the data point onto this axis and association to the nearest class mean. As shown on Figure III-6, the observation plotted would type to the class π_1.

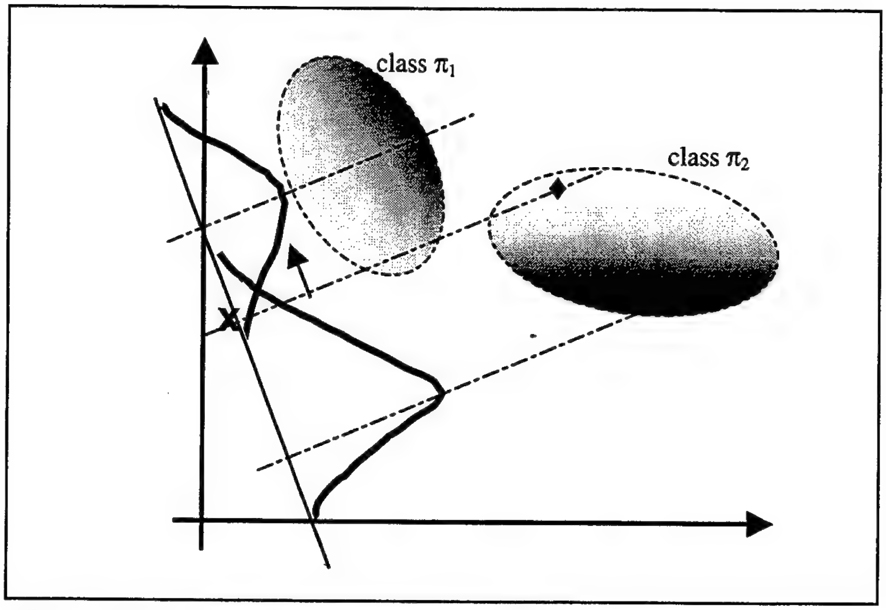


Figure III-6. MSNN Class Typing.

Multiple projection axes are needed to distinguish all pairwise combinations when considering more than two categories. Using the MSNN, Duzenli investigated two methods to identify observations as one of more than two classes. One algorithm determined all possible pairs of classes. For the general case of m classes, namely π_1, π_2, π_3, . . . π_m, k possible combination exist; k determined by

$$k = \binom{m}{2} = \frac{m!}{2!(m-2)!} = \frac{m(m-1)}{2}. \qquad (3.13)$$

Each of the k projections corresponds to a separate processing element in the MSNN.

An alternate classification method suggested by Duzenli separates the data space into class *i* and non-class *i* observations. Segmenting the data as such reduces the required number of processing elements to m, the class number. This second alternative involves a lower computational requirement due to the significantly fewer neurons and, therefore, would appear to be the better choice. Yet, prudence is cautioned when using this latter alternative since assembling the data into class/non-class clusters may alter statistical parameters so as to preclude accurate data typing. Because of this, the strict pairwise routine was followed, irrespective of the higher number of neurons needed. (Duzenli, 1998)

The mechanics of MSNN operations involves three distinct phases: training, typing, and decision-making. Explaining these stages, however, requires understanding the network's basic building block: the MSNN processing element, or neuron. This will be considered next.

2. Processing Element

Shown as Figure III-7, schematically the MSNN processing element differs little from the neuron used in perceptron neural networks. Aside from the inclusion of a scalar

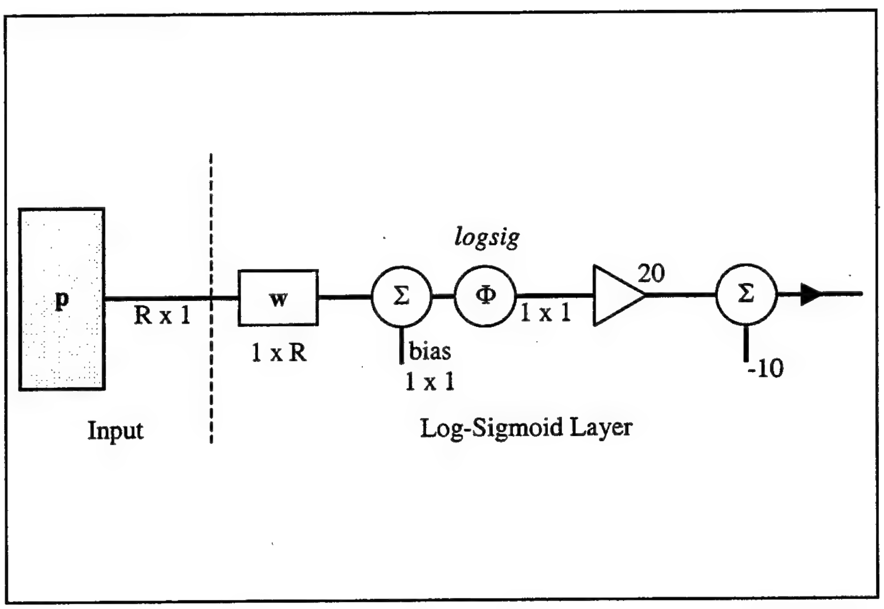


Figure III-7. MSNN Processing Element.

multiplier and adder that serve to increase the neuron's dynamic range by first amplifying and then shifting the activation function output, the principle difference between the perceptron and mean separator processing elements is choice of activation function. Recall that the perceptron uses a hard limit function that maps the neural output to either 0 or 1. Since this transform is not analytic, a principle drawback of the perceptron was that numerical techniques could not be used to optimize a solution.

In contrast, the MSNN does use a differentiable activation function, Φ: the *logarithmic-sigmoid*, or *logsig*, function. The characteristic and closed form equation for the *logsig* function (Figure III-8) define a smooth curve that gradually approaches 1 as its argument increases to positive infinity; and 0, as the argument decreases to negative infinity. Hence, differential optimization methods may be applied to train and improve neuron performance. This network training will be addressed in more detail shortly.

Figure III-7 shows that the MSNN output equals

$$\text{MSNN neuron output} = 20 \cdot \text{logsig}(\mathbf{w} \bullet \mathbf{p} + \text{b}) - 10. \qquad (3.14)$$

As mentioned before, Equation 3.14 incorporates two scalar terms to increase network classification sensitivity. Arbitrarily chosen, the gain value of 20 amplifies the *logsig*

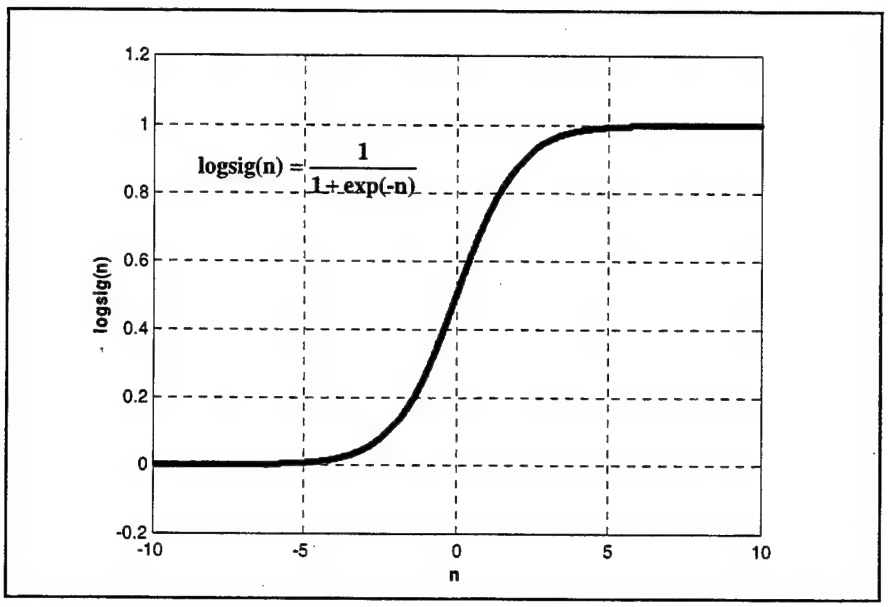


Figure III-8. *logsig* Activation Function.

output while the threshold term sets the MSNN neuron output range at –10 to 10. Implementing this MSNN neural output results in a performance measure and training method that controls weight and bias updates.

3. Training

Equation 3.14 defines the MSNN non-linear transformation. But before this equation can be used for classification, network training is required. This training amounts to determining the projection parameters – that is, the weight vector, **w**, and the bias scalar, b – that maximizes class separation. For the perceptron, these parameters simply defined class boundary lines and were found iteratively by cycling through input data/target output pairs until a specific performance parameter was satisfied. For the MSNN, these weight and bias parameters identify the projection axis upon which maximal mean separation occurs. Consecutive epochs also refine the MSNN parameters, but since the *logsig* activation function is analytic, optimization techniques can be used. This requires identifying a MSNN performance function.

a. Mean-Difference Performance Function

Duzenli defined a *mean-difference* (MD) projection index for the MSNN. This thesis defines an analogous form (Equation 3.15) of his mean-difference equation as:

$$\begin{aligned} MD &= -[E\{(20\cdot\Phi(\mathbf{w}\bullet\mathbf{p}_1+b)-10)-(20\cdot\Phi(\mathbf{w}\bullet\mathbf{p}_2+b)-10)\}]^2 \\ &= -[20\cdot E\{\Phi(\mathbf{w}\bullet\mathbf{p}_1+b)-\Phi(\mathbf{w}\bullet\mathbf{p}_2+b)\}]^2, \end{aligned} \qquad (3.15)$$

with E being the expectation operator and Φ, the *logsig* activation function (Duzenli, 1998). From this equation, the origin of the term "mean-difference" becomes clear. The equation maps observations belonging to two separate classes, denoted by the vectors $\mathbf{p}_1$ and $\mathbf{p}_2$, using the system's performance parameters **w** and b. Applying the non-linear *logsig* function to this linear transformation projects the $\mathbf{p}_1$ and $\mathbf{p}_2$ data spaces onto a one-dimensional projection axis. Taking the difference of the mean of these projections yields the mean-difference.

With regards to Equation 3.15, squaring the mean-difference emphasizes the magnitude, and not the sign, of the difference; while the leading negative sign ensures

upward concavity for function minimization. Recall, from Equation 3.14 that the purpose of the scalar 20 was to increase sensitivity during class typing. Because of this gain, Equation 3.15 gives a mean-difference range of zero (when both data distributions map to 0 or both map to 1) to -400 (when one distribution maps to 1 and the other to 0). The former value correspond to the worse case situation; the latter, to the optimal state.

The MSNN employs supervised, batch processing of input data to train the network. Like a perceptron that undergoes explicit supervised learning in which specific target outputs must be associated with the input data, MSNN learning requires that the training data be assigned to the correct class. As before, batch training refers to parallel processing of the input observations, resulting in a single update per epoch; vice sequential processing in which the system's weights and bias are incrementally changed after each data input. The MSNN training process is schematically shown on Figure III-9 for a three-class classification case.

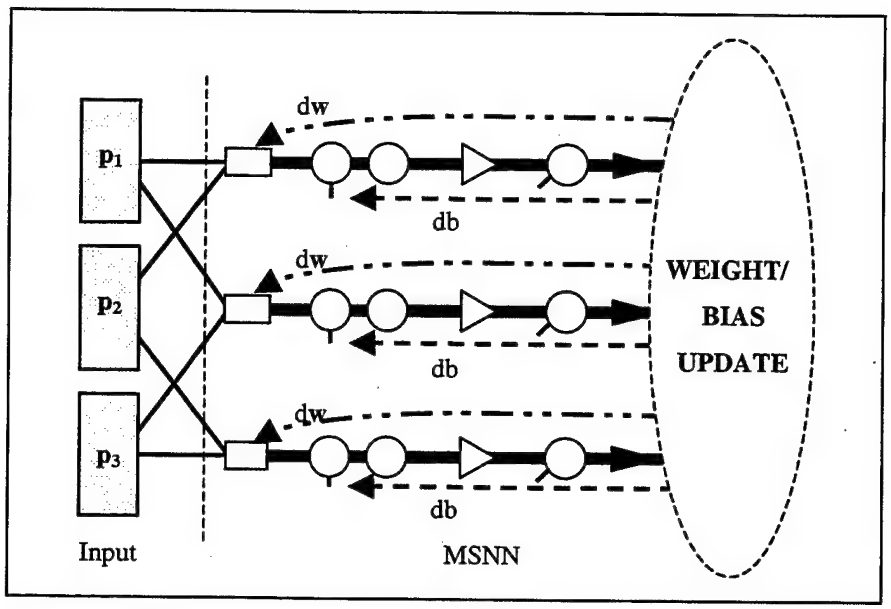


Figure III-9. 3-Class MSNN: Training.

Figure III-9 incorporates three MSNN processing elements into a single layer network. The training process described above prepares the neuro-classifier to recognize classes $\mathbf{p}_1$, $\mathbf{p}_2$, and $\mathbf{p}_3$. Unlike the other phases of MSNN implementation,

during the training stage each neuron simultaneously processes two classes of data, as required by Equation 3.15. The thicker line in the network layer emphasizes this parallel processing. For each neuron, these calculations yield MD values at the input to the "weight/bias update" block. If this value falls below a threshold (empirically determined to be ninety-percent of the optimal value, –360), the neuron's performance parameters require no further training. When the MD value exceeds -360, weight and bias updates, dw and db, are determined using a *steepest descent* algorithm.

b. Weight and Bias Update Equations

When the current projection index is greater than -360, the MSNN parameters update according to equations of the form

$$\mathbf{w}[k+1] = \mathbf{w}[k] + \alpha[k] \cdot \mathbf{f}_1[k] \tag{3.16}$$

$$b[k+1] = b[k] + \alpha[k] \cdot f_2[k], \tag{3.17}$$

where $\alpha[k]\cdot\mathbf{f}_1[k]$ and $\alpha[k]\cdot f_2[k]$ adjust the weight and bias values to improve MD. $\alpha[k]$, a variable learning rate parameter, dictates the incremental step-size towards this upgraded projection index. The analytical meaning of $\mathbf{f}_1[k]$ and $f_2[k]$ are explained next.

For convenience, Equation 3.16 and 3.17 are compacted into a single vector equation:

$$\mathbf{z}[k+1] = \mathbf{z}[k] + \alpha[k] \cdot \mathbf{f}[k]. \tag{3.18}$$

Reiterating that Equation 3.15 drives the weight and bias update, a Taylor's first-order approximation of the mean-difference projection index about a known weight vector and bias yields

$$\mathrm{MD}(\mathbf{z}[k+1]) = \mathrm{MD}(\mathbf{z}[k] + \Delta\mathbf{z}[k]) \approx \mathrm{MD}(\mathbf{z}[k]) + \nabla\mathrm{MD}(\mathbf{z}[k]) \cdot \Delta\mathbf{z}[k], \tag{3.19}$$

with the second term combining the gradient of the performance measure and the change in $\mathbf{z}$. Seeking a trajectory to the optimal MD of -400 and recognizing that this value is also the function's lowest possible value requires that $\mathrm{MD}(\mathbf{z}[k+1]) < \mathrm{MD}(z[k])$. This implies

$$\nabla\mathrm{MD}(\mathbf{z}[k]) \cdot \Delta\mathbf{z}[k] < 0. \tag{3.20}$$

Using Equation 3.18 to define $\Delta\mathbf{z}[k]$ and substituting this into Equation 3.20 results in

$$\alpha[k]\nabla MD(\mathbf{z}[k])\cdot\mathbf{f}[k] < 0, \tag{3.21}$$

with $\alpha[k]$ positive by convention. Since Equation 3.21 is most negative when $\mathbf{f}[k]$ points in a direction opposite that of the gradient, Equation 3.18 becomes

$$\mathbf{z}[k+1] = \mathbf{z}[k] - \alpha[k]\cdot\nabla MD(\mathbf{z}[k]). \tag{3.22}$$

Similarly, Equations 3.16 and 3.17 become

$$\mathbf{w}[k+1] = \mathbf{w}[k] - \alpha[k]\frac{\partial MD[k]}{\partial\mathbf{w}[k]} \tag{3.23}$$

$$b[k+1] = b[k] - \alpha[k]\frac{\partial MD[k]}{\partial b[k]}, \tag{3.24}$$

where the appropriate partial derivative replaces the gradient term. With respect to the weight vector and bias, the partial derivatives of Equation 3.15 are determined to be

$$\frac{\partial MD}{\partial\mathbf{w}} = -800[E\{\Phi(\mathbf{w}\bullet\mathbf{p}_1+b) - \Phi(\mathbf{w}\bullet\mathbf{p}_2+b)\}] \\ *[E\{\Phi'(\mathbf{w}\bullet\mathbf{p}_1+b)\mathbf{p}_1 - \Phi'(\mathbf{w}\bullet\mathbf{p}_2+b)\mathbf{p}_2\}] \tag{3.25}$$

$$\frac{\partial MD}{\partial b} = -800[E\{\Phi(\mathbf{w}\bullet\mathbf{p}_1+b) - \Phi(\mathbf{w}\bullet\mathbf{p}_2+b)\}] \\ *[E\{\Phi'(\mathbf{w}\bullet\mathbf{p}_1+b) - \Phi'(\mathbf{w}\bullet\mathbf{p}_2+b)\}], \tag{3.26}$$

with Φ, the *logsig* activation function, and its derivative shown below:

$$\Phi \equiv \text{logsig}(n) = \frac{1}{1+\exp(-n)} \qquad \Phi' \equiv \text{logsig}'(n) = \frac{1}{\exp(n)\,(1+\exp(-n))^2}.$$

Equations 3.23 and 3.24 comprise the MSNN learning rule. The update terms in these equations correspond to the dw and db terms shown in Figure III-9 that feed back through the neural network. (Hagan, et al, 1996, pp. 9-2 – 9-3)

As an added feature to improve network training, the MSNN step-size, or learning rate, also updates after each iteration. Patterned after the variable learning rate

rules for backpropagation neural networks, the MSNN variable learning rate rules are summarized below (Hagan, et al, 1996, pp. 12-12):

1. If after one epoch the mean-difference parameter increases by more than four-percent (empirically determined), then the trajectory is diverging from the desired state. Consequently, the new weight and bias updates are discarded and the learning rate is halved to minimize movement away from the optimal MD value.

2. If after one epoch the mean-difference parameter increases by less than four-percent, then the trajectory is still diverging from the desired MD value. This movement, however, is tolerable since the change in MD from the previous value is small. For this case, the learning rate is unchanged and the new weight and bias updates are accepted.

3. If after one epoch the mean-difference parameter decreases, then the trajectory is approaching the optimal value. The new weight and bias updates are accepted and the learning rate is doubled to increase movement in this direction.

By doing this, the weight and bias update trajectory are controlled as needed to quickly approach optimal projection index values or to minimize divergence from an acceptable solution.

c. Training Termination

This training scheme updates the MSNN weight vectors and bias values until termination conditions are satisfied; either, the updated MD value is less than the empirically established ninety-percent of optimal (< -360) or a maximum epoch limit is reached. With the network now trained, MSNN classification next involves parameterizing each class to establish the decision rule for separating observations. But, before discussing these subsequent stages, one final point regarding network training must be emphasized. From Figure III-8 (plot of the *logsig* activation function) we recall that the MSNN activation function output asymptotically approaches 0 or 1. The desired solution for a classification problem occurs when one class maps to 0 and the other to 1, as dictated by the argument of the logsig function. Unfortunately, when the initial weight and bias values, instead of the class observation, dominate the output of the linear transform used as the *logsig* argument, the network can become saturated after very little

training. In this saturated state, no further training will occur since the gradient value in these regions is zero. In short, the network has stalled and training will terminate based on the low learning rate (threshold set at 10^{-4}). To prevent this, the network weights and bias are initialized to low magnitude values and the input features are normalized. Hence, network training begins in the sloped region of the *logsig* output to take advantage of this dynamic region and improve the likelihood of satisfactory training.

If training terminates on low learning rate or high epoch cycles and not on acceptable MD, the network is retrained after first discarding and re-initializing the weights and biases. If training ends due to a satisfactory MD level having been reached, the weight and bias values are stored. The MSNN is now ready to proceed to the next phase of determining specific class identifiers.

4. Class Typing and Decision-Making

Tuned to distinguish the different classes, the MSNN must next determine a distinct identifier for each class. Considering a three-class classification problem as before, Figure III-10 diagrams how this is accomplished.

Recall, Figure III-9 showed that the neuron at the top of the diagram (neuron 1) had been trained to separate classes $\mathbf{p}_1$ and $\mathbf{p}_2$. The training data for these two classes

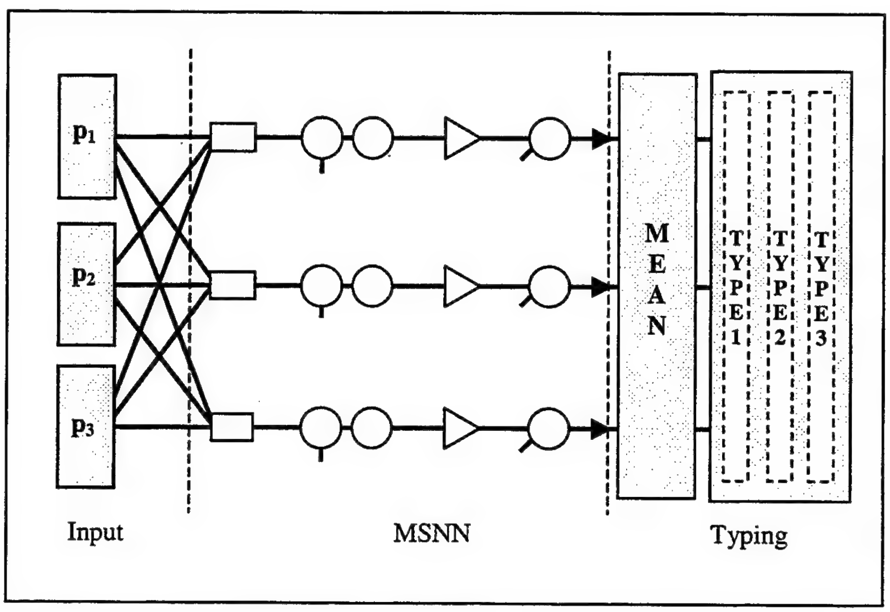


Figure III-10. 3-Class MSNN: Typing.

will again be processed by this neuron. If trained optimally, the processing element will map one class of data to 10 and the other to -10. At the very least, it is hoped the neuron maps one class to a positive value and the other to a negative number. But, should both classes map to the same value after unsatisfactory neuron training, this unfavorable event is not insurmountable. Since the data point mappings from all neurons comprise the class identifier, even if one processing element is poorly trained, the other neurons may potentially provide for unique class identifiers.

For now, however, assume a $\mathbf{p}_1$ data point generates 10, while a $\mathbf{p}_2$ observation turns out -10. A class $\mathbf{p}_3$ data point will also be cycled through neuron 1, resulting in another -10, for instance. Consequently, after taking one observation from each class and mapping them by neuron 1, the following distinction shown as Table III-2 is realized:

		CLASS $\mathbf{p}_1$	CLASS $\mathbf{p}_2$	CLASS $\mathbf{p}_3$
NEURON 1	1,2	10	-10	-10

Table III-2. Hypothetical Class p1, p2, and p3 Output from Trained Neuron 1 (Class p1 vs Class p2).

In Table III-2, the second column indicates the two classes used to train the neuron.

Using the same three training data points, output from the remaining two neurons are also determined. Completing Table III-2 with these remaining data points shows the unique identity of each class type.

		CLASS $\mathbf{p}_1$	CLASS $\mathbf{p}_2$	CLASS $\mathbf{p}_3$
NEURON 1	1,2	10	-10	-10
NEURON 2	1,3	10	-10	-10
NEURON 3	2,3	10	-10	10

Table III-2a. Hypothetical Class p1, p2, and p3 Output from Trained 3-Class Neural Network.

Notice that if neuron 1 had mapped the data points from all classes to 10, for this example the three classes would still have unique identifiers. In general, however, this is not true. Neurons 2 and 3 could have been trained such the resulting specifiers did not uniquely identify each class type.

When determining class specifiers, the network does not process only one point from each class through the neurons. To obtain a representative template for each class, the trained neural network processes all training data. This produces a neuron map of all data points as shown as Figure III-11. Calculating the average output from each neuron for each class determines the three class specific identifiers. These identifiers, $\mathbf{r}_1$, $\mathbf{r}_2$, and $\mathbf{r}_3$ in the three-class case, are then saved for later use in classifying observations.

Up to this point the MSNN has processed only training data. Once the network has learned the characteristics of the input data and can distinguish the separate classes, it can be used to classify new observations. Shown schematically on Figure III-12, this

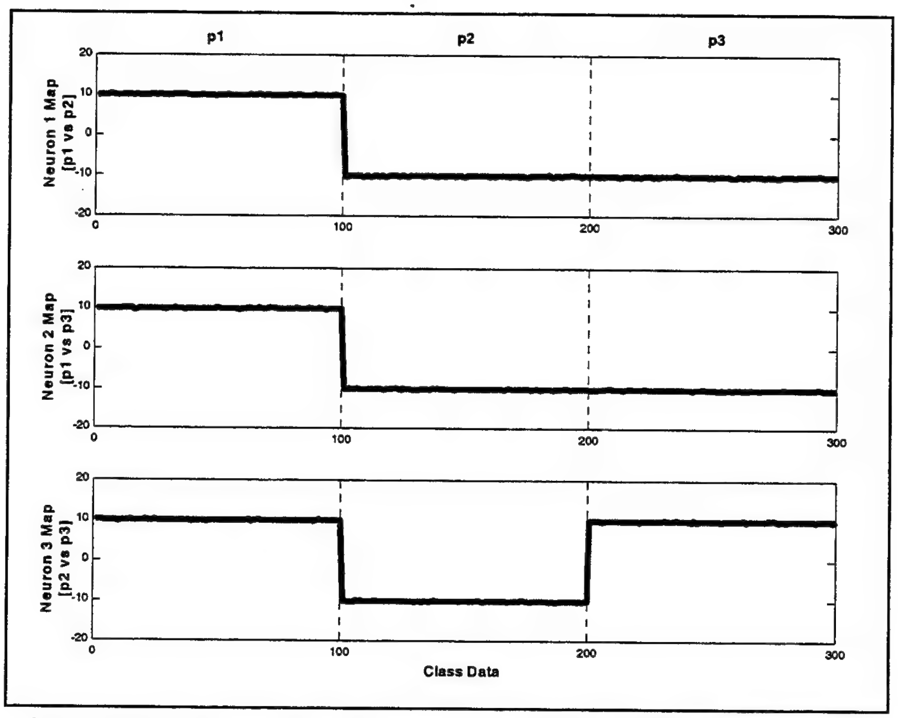


Figure III-11. Neuron Maps for Hypothetical 3-Class MSNN Typing. Each plot depicts how a trained neuron maps class data. Read vertically, the plots identifies the unique class type specifiers produced by the MSNN.

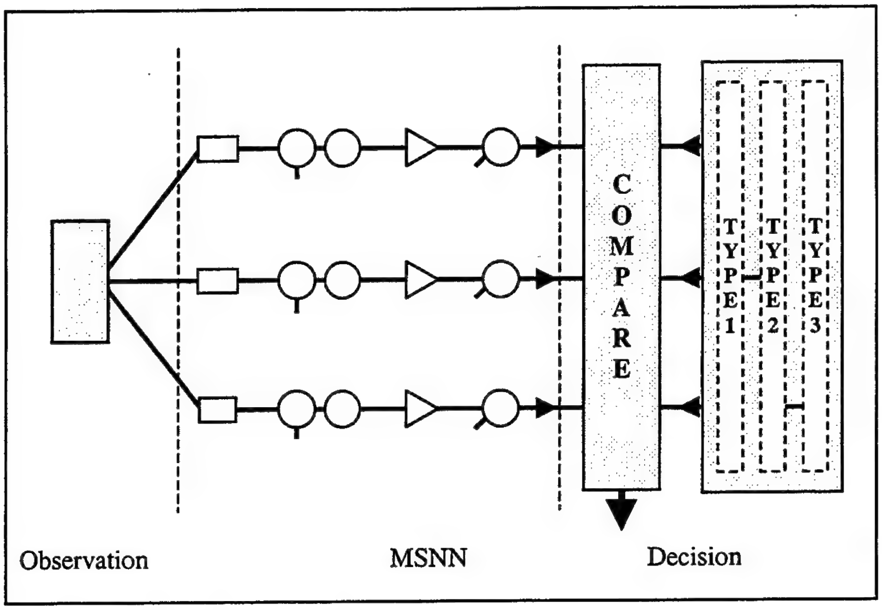


Figure III-12. 3-Class MSNN: Decision-Making.

process comprises the final stage in classifying observations with the MSNN: decision-making.

The decision phase begins when a sensor or data storage device provides the tuned MSNN with an observation. Needless to say, if the training data was conditioned prior to being processed by the MSNN, so must this new observation. According to Equation 3.14, the MSNN maps this observation producing an output from each neuron. This observation typing, $\mathbf{o}$, is compared to the stored class specifiers, $\mathbf{r}_i$, via an Euclidean distance measurement of the general form

$$d_i = (\mathbf{r}_i - \mathbf{o}_i)^T \bullet (\mathbf{r}_i - \mathbf{o}_i) \qquad \text{for } i = 1, 2, \ldots, m \tag{3.27}$$

with the index i indicating a particular class. The minimum distance measure associates the observation to a particular class.

5. Summary

Summarizing the main MSNN principles, this section has shown:

1. The MSNN projects observations onto the one-dimensional axis that maximizes separation between the mean value of two class clusters.

2. The MSNN processing elements utilize a differentiable activation function (*logsig*) that saturates at 1 and 0 for input arguments of positive infinity and negative infinity, respectively. Optimal performance requires initialization of the network weight and bias to low values to prevent early network saturation at these asymptotic values.

3. The MD optimal value of -400 is attained when one class maps to 10 and the other to -10. The worse case MD value of 0 occurs when the two classes type to the same output value (both classes mapping to either 10 or -10).

4. The MSNN training follows a steepest descent algorithm that incorporates a variable learning rate and terminates when ninety-percent of the optimal MD value is reached. Short of attaining this, MSNN training will cease when the learning rate falls below a set lower limit or when a maximum number of training epochs is achieved. If either of these latter cases were to occur, the weights and bias would be discarded and re-initialized for re-training.

5. Once trained, the MSNN processes the training data to determine specific class identifiers.

6. When available, a new observation is processed through the trained MSNN. The projection of this observation by the neural network is compared to the class identifiers. Using an Euclidean distance measure, the observation is associated with a class.

Previous trials have demonstrated the classification capabilities of the MSNN (Duzenli, 1998). As indicated above, this was accomplished by training the neural network to maximize the separation between the projected means of two class clusters. Relying on maximal mean separation, however, may not adequately ensure minimal cluster overlap and, hence satisfactory classification performance. The next section expounds on the reasons for this behavior and suggests modification to the mean separator classification scheme.

D. ALTERNATE MEAN SEPARATOR SCHEMES

Repeated here, Figure III-5 illustrates the principle purpose of the MSNN. As previously explained, the original MSNN algorithm favors case (a) because of the larger spread between projected cluster means. Yet, examination of this choice demonstrates an incongruity of the standard MSNN process. Although case (a) does display greater mean separation, more cluster overlap also occurs with this selection of projection direction.

Consequently, an observation belonging to class π_2 may type to class π_1, an inaccurate selection, because of its position relative to the data cluster. For this reason, case (b) would be more appropriate. Figure III-13 illustrates this situation.

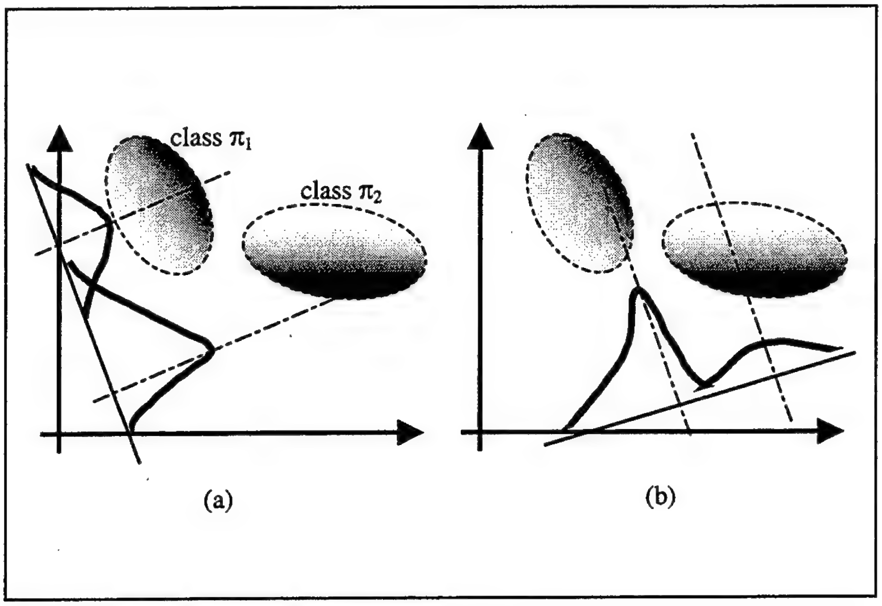


Figure III-5 (repeated). MSNN Projection.

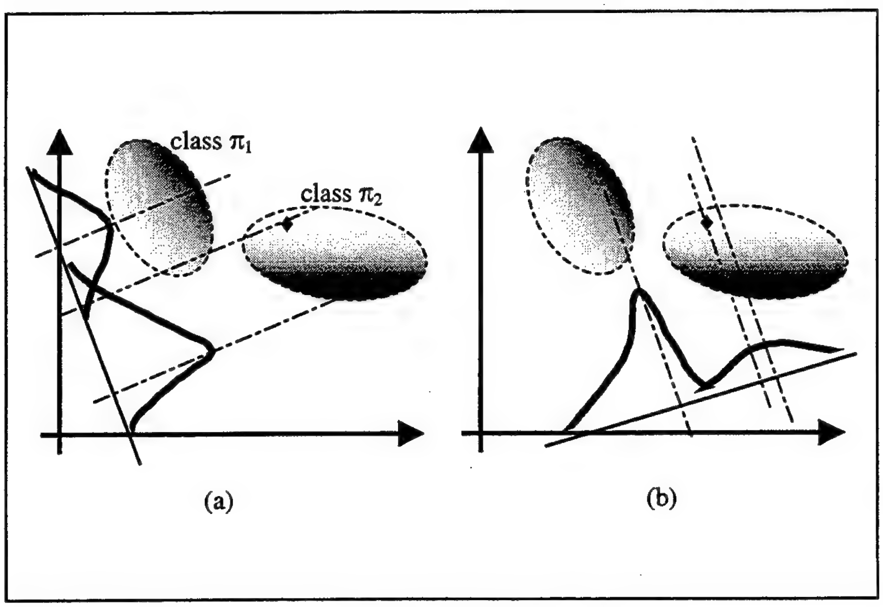


Figure III-13. Anomalous MSNN Classification Situation.

Ironically, the effect of such a situation would be more profound when there are fewer class choices. Recall that the number of class alternatives determines the network size. Fewer possibilities result in a network consisting of a diminished number of processing elements. This would be disadvantageous since the effect of the irregularity shown in Figure III-13 could not be offset by the increased network flexibility provided by other neural mappings. Fortunately, the typical classification situation would entail more than a few possible choices, so the likelihood of this scenario would be minimal. Moreover, techniques that compensate for data variance can prevent erroneous classification such as this. Three such methods are explained here. The first adjusts the MSNN classification scheme by pre-processing the input data. The second alteration normalizes the class spread by considering projected data variance. Finally, the third applies a termination parameter defined for the second modification method to the standard MSNN.

1. Input Data Preconditioning

The first attempt to counter overlapping projections of two different classes involves normalizing the input data distribution. It was conjectured that a tighter data spread would effect smaller group projections, thereby facilitating class separation. Figure III-14 demonstrates this hypothesis.

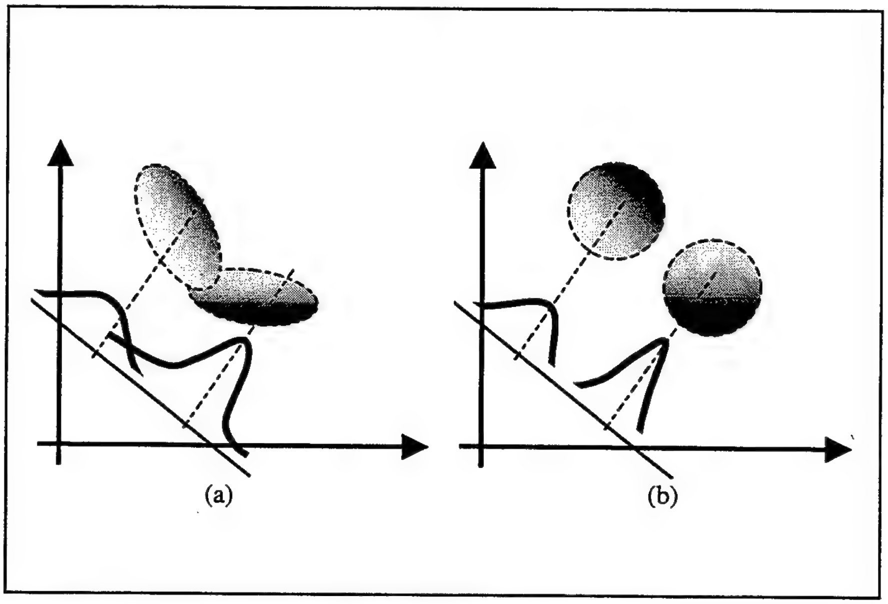


Figure III-14. Postulated Effect of Data Preconditioning.

With this data pre-processing approach, changes to MSNN training and typing algorithms are not needed. However, in addition to the required preconditioning of training data and observations, a more sophisticated decision-making scheme would be implemented.

Prior to submitting training data to the MSNN, the training data is normalized according to

$$\mathbf{p}_i^* = \frac{(\mathbf{p}_i - \boldsymbol{\mu}_i)}{\boldsymbol{\sigma}_i} + \boldsymbol{\mu}_i, \tag{3.28}$$

with $\mathbf{p}_i$ and $\mathbf{p}_i^*$ respectively being the data values before and after normalization; $\boldsymbol{\mu}_i$ representing a vector of class feature mean values; and $\boldsymbol{\sigma}_i$ representing a vector of class feature standard deviation values. We recognize that this normalization preserves the mean values by removing the feature averages and then reapplying them after scaling. With n training data points and m classes, training data normalization would increase the number of floating point operations by a factor of n∗m.

Having been trained with normalized data, for the MSNN to accurately classify uncategorized data the observations must be similarly adjusted. Therefore, Equation 3.28 is also applied to unclassified observations prior to processing by the MSNN. But while the training data can be associated to a particular class, the nature of the classification problem dictates that the class of the observation is obviously unknown. Preconditioning of observations consequently calls for data normalization by the statistical parameters of all possible classes. Accordingly, the computational requirement has been increased by a factor of m, the number of classes.

Using the adjusted training data, the MSNN's performance parameters and class identifiers are determined, as described previously by Figures III-9 and III-10. All equations used during the MSNN training and typing phase apply. The trained network then transforms the normalized observations into the decision space, where the network compares each mapped outcome to the identifier of the particular class associated with that scaled version. That is, the output resulting from an observation scaled by class *i* statistics would be compared to the class *i* type identifier. In the end, the class identifier

most similar to its corresponding network output as determined by Euclidean distance is chosen as the proper category of the observation. Compared to that of the standard MSNN classifier, each mapping and matching routine entails no additional computations. True, each observation would undergo m such processes, one for each observation scaling; but, this factor has already been justified. Overall then, an input preconditioning approach increases the number of computer operations by a factor of $(n+1)*m$. For large training sets and many distinct classes, the added computational load is not trivial.

Yet despite this drawback, the disadvantage caused by a large computational requirement could be overlooked if actual trials demonstrate a considerable improvement in network performance. Unfortunately, enhanced robustness may not be demonstrated when input standard deviations are less than one. Under these conditions, normalization would make the training data distributions more diffuse and not compact. In addition, since the normalization is performed in the feature space, the effect of input data preconditioning may not affect the decision space as positively as Figure III-14 shows. The mapping of the normalized data points may cause the projection distributions to be tighter, more spread out, or unchanged depending on the neural networks initializations and training trajectory. For these reasons, decision space normalization is considered as a second method to enhance MSNN performance.

2. Projection Space Normalization

a. Concept

By reducing the feature space noise level, the first modification to the MSNN classification scheme sought to improve network performance with only minimal changes to the standard algorithm. Believing input data normalization would result in a less ambiguous, more tightly clustered class distribution, it was thought projection into the MSNN decision space would not disrupt this cohesion. Consequently, the resulting compact clusters would enhance class separation.

Upon reconsideration, however, it was recognized that (1) normalization may not reduce the variance of the data distribution (e.g., in case in which the feature standard deviation was already less than one) and (2) since the MSNN transformation is

non-linear, projection into the decision space could detrimentally alter the data distribution within a cluster.

So, instead of trying to obtain an optimal output by pre-processing the input features, a second variation of the MSNN would instead optimize the output obtained. By minimizing the variance of the projected data while still maximizing mean separation, projection cluster overlap would be reduced, thereby lowering the likelihood of inaccurate classification. As a result of this combination of actions, a large variance may be tolerable if mean separation is likewise large; while a smaller spread could be unacceptable for closely spaced class groupings. Figure III-15 illustrates this notion.

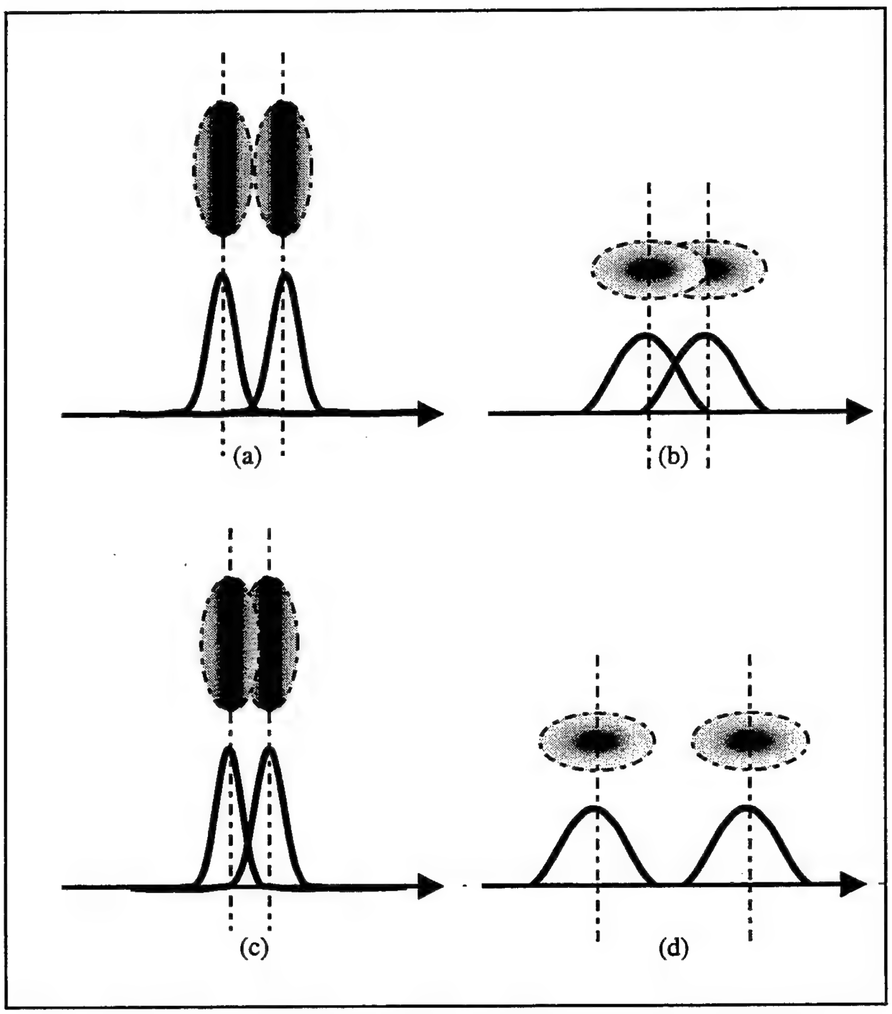


Figure III-15. Relative Significance of Mean Separation to Variance.

Shown in the decision space, Figure III-15 illustrates four combinations of mean separation and variance and the resulting effect on classification capabilities. For instance, plots (a) and (b) illustrate the obvious conditions with respect to distribution variance. For a given mean separation, overlap is unlikely with low data spread (plot (a)); while the converse is true with large variance (plot (b)). Figures III-15 (c) and (d), however, emphasize that it is the relative, and not absolute, magnitudes of mean separation and variance that are significant. In plot (c), large overlap occurs despite low variance; but in plot III-15(d), no overlap results regardless of a large variance. Therefore, the approach does appear to be more logical than either of the two earlier MSNN models.

Executing this process, however, will involve changes to the MSNN procedure. The MSNN class typing and decision-making phases depicted in Figures III-10 and III-12 are still applicable and will not require change; but aspects of the training phase will need revision. Alterations to the training performance measure and the training termination criteria are considered.

b. Modified Mean-Difference Projection Index

MSNN training with projection space normalization does not require modification to the network training procedure. The processing element and the data flow path as depicted earlier in Figures III-7 and III-9 remain unchanged. The performance measure specified by Equation 3.15, however, will be modified. Taking into consideration the projection space variance of the two transformed data distributions, the new mean-difference projection index (MD_2) is defined as

$$\begin{aligned}MD_2 &= -\frac{[E\{(20\cdot\Phi(\mathbf{w}\bullet\mathbf{p}_1+b)-10)-(20\cdot\Phi(\mathbf{w}\bullet\mathbf{p}_2+b)-10)\}]^2}{var(20\cdot\Phi(\mathbf{w}\bullet\mathbf{p}_1+b)-10)+var(20\cdot\Phi(\mathbf{w}\bullet\mathbf{p}_2+b)-10)}\\ &= -\frac{[E\{\Phi(\mathbf{w}\bullet\mathbf{p}_1+b)-\Phi(\mathbf{w}\bullet\mathbf{p}_2+b)\}]^2}{var(\Phi(\mathbf{w}\bullet\mathbf{p}_1+b))+var(\Phi(\mathbf{w}\bullet\mathbf{p}_2+b))},\end{aligned} \tag{3.29}$$

where Φ again represents the *logsig* activation function and var symbolizes the statistical variance.

Because of this new projection index, the gradient portion of the mean-difference learning rate must be recomputed. Taking the partial derivatives of MD_2, as specified by Equations 3.23 and 3.24, yields

$$\frac{\partial MD_2}{\partial \mathbf{w}} = 2K[K(E\{\alpha\frac{\partial \alpha}{\partial \mathbf{w}} + \beta\frac{\partial \beta}{\partial \mathbf{w}}\} - E\{\alpha\}\cdot E\{\frac{\partial \alpha}{\partial \mathbf{w}}\} - E\{\beta\}\cdot E\{\frac{\partial \beta}{\partial \mathbf{w}}\}) - E\{\frac{\partial \alpha}{\partial \mathbf{w}} - \frac{\partial \beta}{\partial \mathbf{w}}\}] \quad (3.30)$$

$$\frac{\partial MD_2}{\partial b} = 2K[K(E\{\alpha\frac{\partial \alpha}{\partial b} + \beta\frac{\partial \beta}{\partial b}\} - E\{\alpha\}\cdot E\{\frac{\partial \alpha}{\partial b}\} - E\{\beta\}\cdot E\{\frac{\partial \beta}{\partial b}\}) - E\{\frac{\partial \alpha}{\partial b} - \frac{\partial \beta}{\partial b}\}], \quad (3.31)$$

with the parameters K, α, and β defined as

$$K = \frac{E\{\alpha - \beta)}{E\{\alpha^2 + \beta^2\} - E^2\{\alpha\} - E^2\{\beta\}}$$

$$\alpha = \Phi(\mathbf{w}\bullet\mathbf{p}_1 + b), \frac{\partial \alpha}{\partial \mathbf{w}} = \Phi'(\mathbf{w}\bullet\mathbf{p}_1 + b)\cdot\mathbf{p}_1, \frac{\partial \alpha}{\partial b} = \Phi'(\mathbf{w}\bullet\mathbf{p}_1 + b)$$

$$\beta = \Phi(\mathbf{w}\bullet\mathbf{p}_2 + b), \frac{\partial \beta}{\partial \mathbf{w}} = \Phi'(\mathbf{w}\bullet\mathbf{p}_2 + b)\cdot\mathbf{p}_2, \frac{\partial \beta}{\partial b} = \Phi'(\mathbf{w}\bullet\mathbf{p}_2 + b).$$

As before, the *logsig* activation function, Φ, and its derivative are defined by

$$\Phi \equiv \text{logsig}(n) = \frac{1}{1+\exp(-n)} \qquad \Phi' \equiv \text{logsig}'(n) = \frac{1}{\exp(n)(1+\exp(-n))^2}.$$

Note that MD_2, α, β and their derivatives with respect to the neural network bias are all scalar quantities. The derivatives of these parameters with respect to the weight vector are, on the other hand, vectors. This agrees with the MSNN learning rule equations, Equations 3.23 and 3.24.

With the projection index now expressed as a ratio of mean separation to sum of projection variance, the range is no longer constrained to [-400,0]. In fact, in the optimal situation, the sum of variance is zero and therefore MD_2 is undefined. Conceptually, a small variance and the resulting large magnitude for MD_2 concurs with the best case situation described by the numerator of the projection index, that of a large mean difference. But, an infinitesimally small denominator causes computational difficulties. To preclude this, the denominator of MD_2 and its derivatives are limited to a minimum value of 10^{-10}.

c. *Modified Termination Requirement*

The training phase of the standard MSNN terminated either on maximum epoch limit, minimum learning rate, or optimal performance measure. The first two criteria are still valid within the framework of the projection space variance modification; however, the latter case no longer has any meaning. In the best case scenario, the performance measure is unbounded and thus cannot be used to end training. Multiplying the MD_2 projection index by its denominator (i.e., the sum of projection variances) may allow for implementation of a termination criteria; but, this termination requirement would amount to only the projection space mean separation, thereby ignoring the relevance of data spread. Because of this, a new termination index that measured the ratio of data variance to mean separation was defined.

Consider the projection space data distributions shown on Figure III-16. Improving classification performance relies on maximizing the separation, ΔV, between the points x_1 and x_2 relative to the mean separation, ΔM. Based on an error tolerance, these points are found using statistical error function tables, assuming both projected data sets are normally distributed. The termination parameter, the variance-mean ratio (VMR), is then defined as $\Delta V/\Delta M$. For a given mean separation, imposing a threshold on this ratio specifies the minimum spread value ΔV and consequently the allowed variance of the projected class distributions. A more rigorous derivation of this parameter follows.

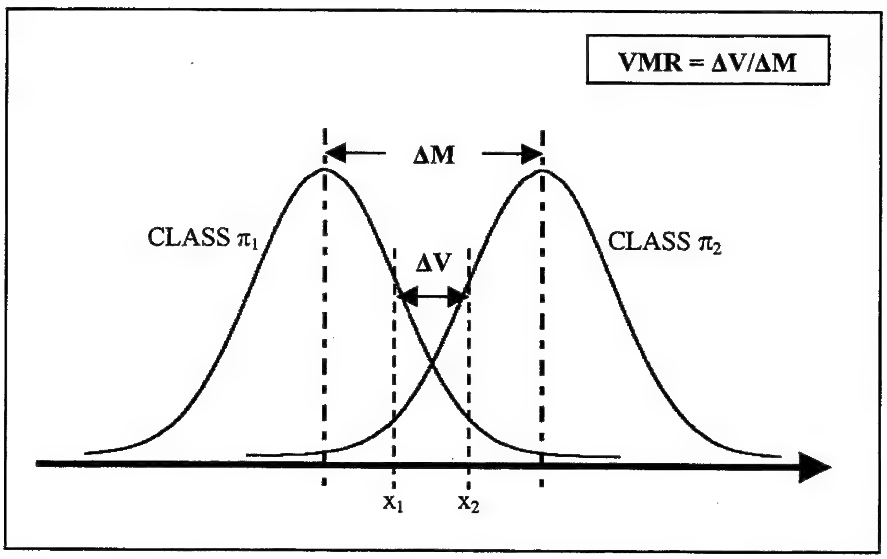


Figure III-16. Variance-Mean Ratio.

The primary assumption needed for the derivation of the VMR criterion is that, in the decision domain, the projected data is normally distributed. By making this claim, error function tables and known characteristics of normal distributions can be used to analytically derive VMR. But, to verify this supposition requires examining the attributes of the projected data. Figures III-17 through III-20 illustrate the transformed data distributions for each class of a two-class classification problem. Plots (a) and (b) display the normality plots of the resulting distributions. A non-vertical, linear plot of '+' marks superimposed on the dashed line denotes a Gaussian distributed data set. In contrast, a curvature in the plotting of these marks indicates a departure from normality. Plots (c) and (d) are the corresponding histograms.

In the optimal case (Figure III-17), the data is far from Gaussian. This, however, is desired. Instead of the expected bell-shaped data distribution characteristic of a Gaussian curve, the data shown in Figure III-17 shows one vertical bar. Recall that when optimally trained, the MSNN processing element will precisely map one class to 10 and the other –10, as shown. As will be defined shortly, VMR for this case is 1 and the assumption of normality is not required.

In the least desired situation depicted by Figure III-18, the data is again far from Gaussian. Although two vertical bars are now shown for each class, indicating poor data classification, all mappings are precisely to one of the extreme values. Consequently, mapping into the projection space did not result in data overlap and the assumption of normality is again not required.

In the intermediate cases shown on Figures III-19 and III-20, it is apparent that the transformation into the decision space was not precise. As a result data overlap may occur. In three of the four cases shown (both classes of Figures III-19 and class π_2 of Figure III-20) the distributions are nearly normal, so the initial assumption holds. For class π_1 of Figure III-20, however, the normality plot indicates that the tail of the distribution extends further out than that of a normally distributed data set. This implies a greater amount of data overlap than assumed by a Gaussian distribution. Fortunately, this situation is atypical. Because of the *logsig* activation function, the input data tends to

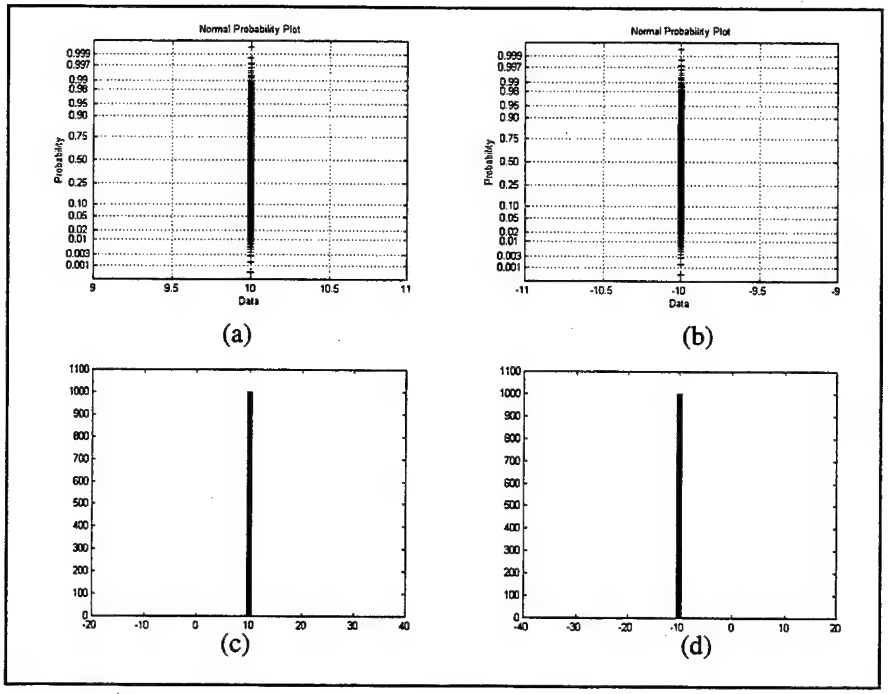


Figure III-17. Example of Projected Data Distribution. (a) Class π_1 Normality Plot (b) Class π_2 Normality Plot (c) Class π_1 Histogram (d) Class π_2 Histogram.

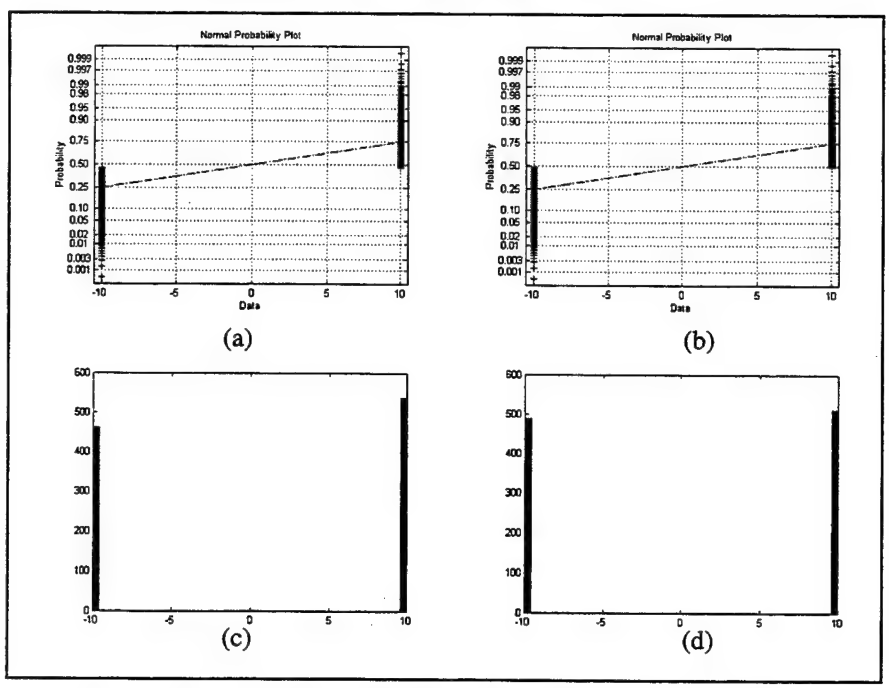


Figure III-18. Example of Projected Data Distribution. (a) Class π_1 Normality Plot (b) Class π_2 Normality Plot (c) Class π_1 Histogram (d) Class π_2 Histogram.

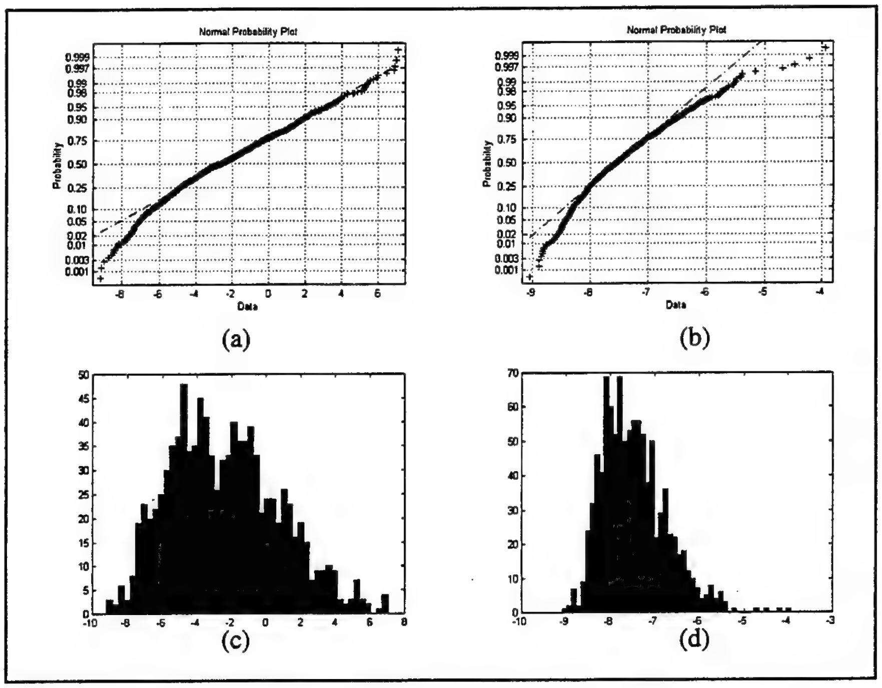


Figure III-19. Example of Projected Data Distribution. (a) Class π_1 Normality Plot (b) Class π_2 Normality Plot (c) Class π_1 Histogram (d) Class π_2 Histogram.

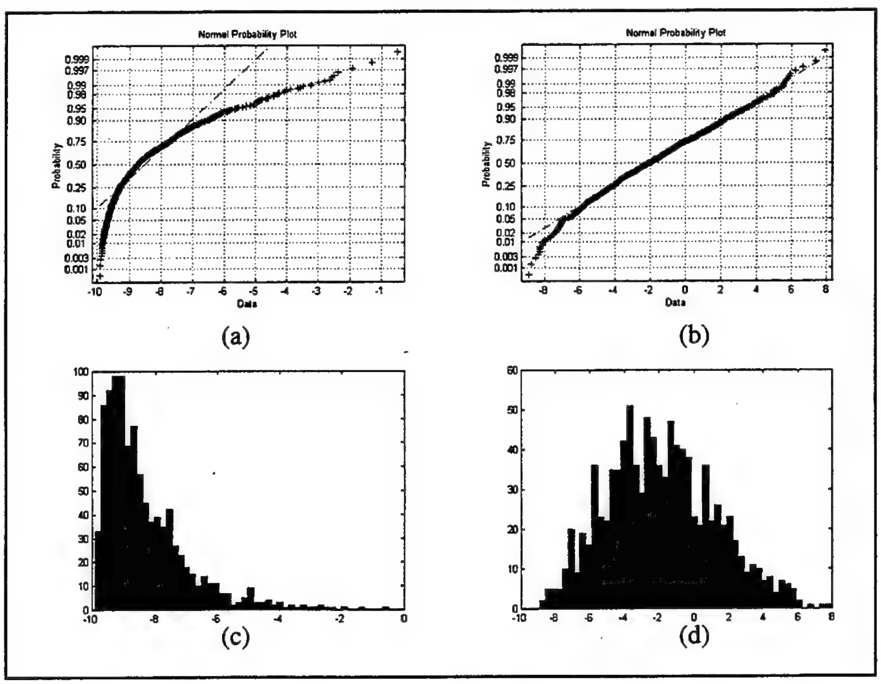


Figure III-20. Example of Projected Data Distribution. (a) Class π_1 Normality Plot (b) Class π_2 Normality Plot (c) Class π_1 Histogram (d) Class π_2 Histogram.

map to one of the optimum values (i.e., 10 or -10). Yet, to compensate for this aberrant case, stringent requirements will be placed on VMR.

Accepting the assumption of a normally distributed data projection, the derivation for VMR is as follows. For the two classes shown in Figure III-16, the projection of class π_1 has a mean μ_1 and standard deviation σ_1. Correspondingly, the projection of class π_2 has a mean μ_2 and standard deviation σ_2. Unlike in the feature space, the class means and standard deviations are scalar quantities owing to the one-dimensional projection by the neural network's linear and non-linear mappings.

Taken from error function tables, the error tolerance specifies the location of x_1 and x_2 on the projection axis. For instance, with an allowable error set at 0.5%, the threshold points for a zero-mean, unit-variance, normally distributed class are ±2.52 units from the mean. That is, 0.5% of the distribution reside in the tails beyond these locations. Applying the known statistical parameters of the actual classes, these positions are found to be

$$x_a = \mu_a + 2.52 \cdot \sigma_a, \tag{3.32}$$

$$x_b = \mu_b + 2.52 \cdot \sigma_b. \tag{3.33}$$

In Equations 3.32 and 3.33, the subscripts a and b are used to derive the formulae without having knowledge of the actual orientation of classes π_1 and π_2. In the general sense, subscript b refers to the class with the more positive mean. So, in terms of Figure III-16, x_a corresponds to x_1; x_b to x_2. Taking the difference of x_b and x_a yields ΔV:

$$\begin{aligned} \mathbf{\Delta V} &= x_b - x_a \\ &= (\mu_b - 2.52 \cdot \sigma_b) - (\mu_a + 2.52 \cdot \sigma_a) \\ &= (\mu_b - \mu_a) - 2.52(\sigma_b + \sigma_a). \end{aligned} \tag{3.34}$$

Using Equation 3.34, the variance-mean ratio (VMR) can be expressed as

$$\begin{aligned} \mathrm{VMR} &= \frac{\Delta V}{\Delta M} = \frac{(\mu_b - \mu_a) - 2.52(\sigma_b + \sigma_a)}{\mu_b - \mu_a} \\ &= 1 - \frac{2.52(\sigma_b + \sigma_a)}{\mu_b - \mu_a}. \end{aligned} \tag{3.35}$$

To account for cases in which improper class assignment results in the mean of class a being more positive than the mean of class b, an absolute value is introduced to emphasize the magnitude and not the sign of the difference in means. Equation 3.35 therefore becomes

$$\mathrm{VMR} = 1 - \frac{2.52(\sigma_b + \sigma_a)}{|\mu_b - \mu_a|}. \tag{3.36}$$

If Equation 3.36 had been incorrectly derived, the second term would have been added instead of subtracted.

Equation 3.36 establishes how tightly clustered the class projection into the decision space must be. Recognizing that a VMR of zero would only incur the acceptable error limit (here, 0.5% error) for a Gaussian distributed data sample, a VMR greater than zero imparts an even higher requirement on projected class variance. This compensates for any situations in which the data distribution is not Gaussian and institutes the precision required of the neural network training. Caution must be observed for negative VMR values. This implies a mean separation that is smaller than the sum of variances and hence, a large degree of overlap.

During actual implementation, VMR terminated the training cycle only after an improvement in MD_2 (i.e., a more negative value). In retrospect, however, checking MD_2 was not required. Since this modification considers both mean separation and projection variance, an increase in mean-difference (MD_2) does not necessarily indicate worsening conditions, as it does for the mean-difference (MD) of the standard MSNN. Consequently, network training should have been stopped on VMR threshold, maximum epoch limit, or minimum learning rate, without consideration for the MD_2 projection index.

3. Further Implementation of the Variance-Mean Ratio

Perhaps the strength of projection space normalization modification does not lie in the upgraded performance parameter, MD_2, as originally intended, but rather in the termination parameter, VMR. Because of this possibility, the third MSNN variation used

VMR, vice the empirically determined ninety-percent of optimal MD, as the training termination requirement for the original MSNN method.

E. SUMMARY

Chapter III discussed several techniques used to classify observations. These methods include a parametric statistical classifier and five neural network architectures. The statistical classifier of interest was a quadratic classifier. The decision rule for this method was derived and its applicability to normally distributed data, highlighted.

The first neural network examined was the single layer perceptron. This neuro-classifier used linear separation boundaries to partition classes into their own separate spaces. The primary difficulty encountered with the perceptron networks was the inability to use optimization techniques to guide the network's training. Instead a simple, albeit powerful under certain situations, rule governs perceptron learning.

Next, the Mean Separator Neural Network (MSNN) first introduced by Duzenli and Fargues was explained. This network architecture and variations on its design are the principle focus of this study. Classification with MSNN are performed by projecting data onto an one-dimensional axis. The mean-difference (MD) performance parameter maximizes the separation between class mean values, enabling classification of observations to the proper category by using a distance metric.

Improved performance was sought by modifying the MSNN to consider the data variance. One alternative mean-separator normalized the input space in an attempt to produce tight class clusters. A second, more promising, approach normalized the projection space using an upgraded performance parameter, MD_2, and a new training termination criteria, VMR. Together, these metrics maximized the projected mean separation while also tightening the decision space data spread, reducing data cluster overlap. Hypothesizing, however, that the primary driver to restricting this overlap was the termination parameter, VMR, and not the modified performance parameter, MD_2, classification using the standard MSNN projection index, MD, coupled with the new termination criteria was considered as a third modification to the MSNN. In the following chapters, these MSNN variants – MSNN with preconditioned input space,

MSNN with normalized projection space, and MSNN with VMR termination – will be referred to as MSNN Mod 1, MSNN Mod 2, and MSNN Mod 3, respectively.

Chapter IV will next discuss the preliminary investigations into the effectiveness of the individual classification tools.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. VERIFICATION OF CLASSIFIER PERFORMANCE

Chapter III introduced and explained the implementation of the different classifiers considered in this study; one parametric classifier and five neural networks. This chapter assesses these methods through simulations. MATLAB program codes used during these trials are provided in Appendix C.

A. SIMULATION PROTOCOL

A three-class separation problem was considered to test the performance of the subject classification methods. Working in three-, ten-, and fifty-dimension input spaces, the classifiers used 100 training objects per class to model the data and then used this representation to categorize 1000 trial observations per class. Performing the tests under various noise conditions emphasized the robustness of the classification methods. Specifically, the signal-to-noise ratios (SNRs) simulated were ±20 dB, ±15 dB, ±10 dB, ±5 dB, and 0 dB. Absent from this list is the no-noise case since generation of zero-variance data would identify only one point for each class.

Constructing the training and testing data objects required determining class statistics. The mean values for each class feature were randomly selected from a uniform distribution. To focus the initial neural network activity in the *logsig* dynamic range and thereby prevent neural network saturation, these mean values were constrained to [-1,1]. During real-time analysis, signal power is normalized. Hence, the normalized sum of n feature variances gives signal SNR, as shown by Equation 4.1:

$$\mathrm{SNR} = 10\log_{10}\left(\frac{n}{\sigma^T \bullet \sigma}\right). \qquad (4.1)$$

Consequently, when SNR is known Equation 4.1 can be used to randomly select each feature variance from a uniform distribution.

Having randomly specified the mean and established the variance values for each class, Gaussian distributed features were simulated to form the 300 training and 3000 testing observations (100 and 1000 for each class) required per trial. Examples of a

three-class, three-feature classification problem with low and high noise conditions are illustrated in Figure IV-1 and IV-2. The two-dimensional plots in each figure depict data projection onto two of the three dimensions. As expected, decreased SNR resulted in increased data overlap, thereby suggesting increased classification difficulty.

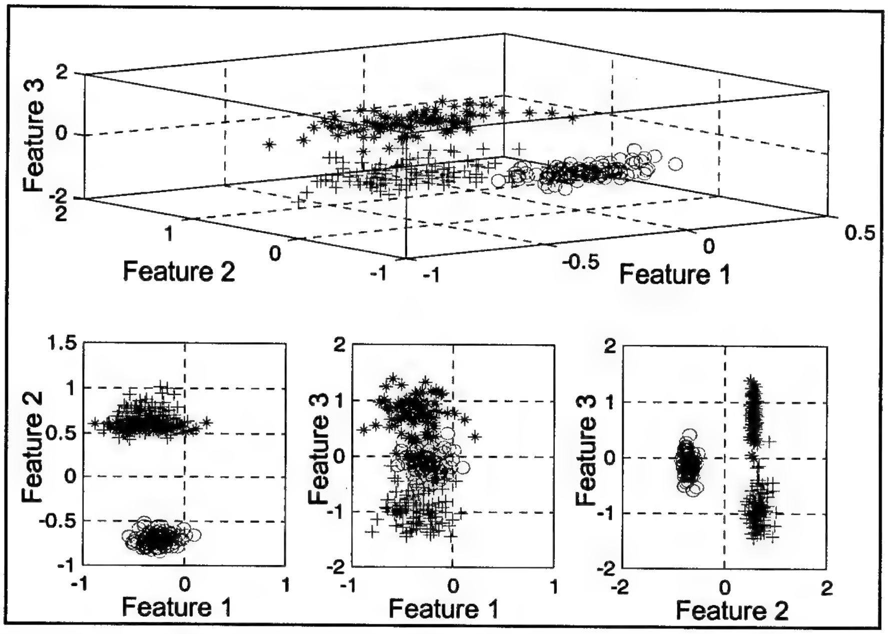


Figure IV-1. Example of 3-Feature Data for Classification (low noise).

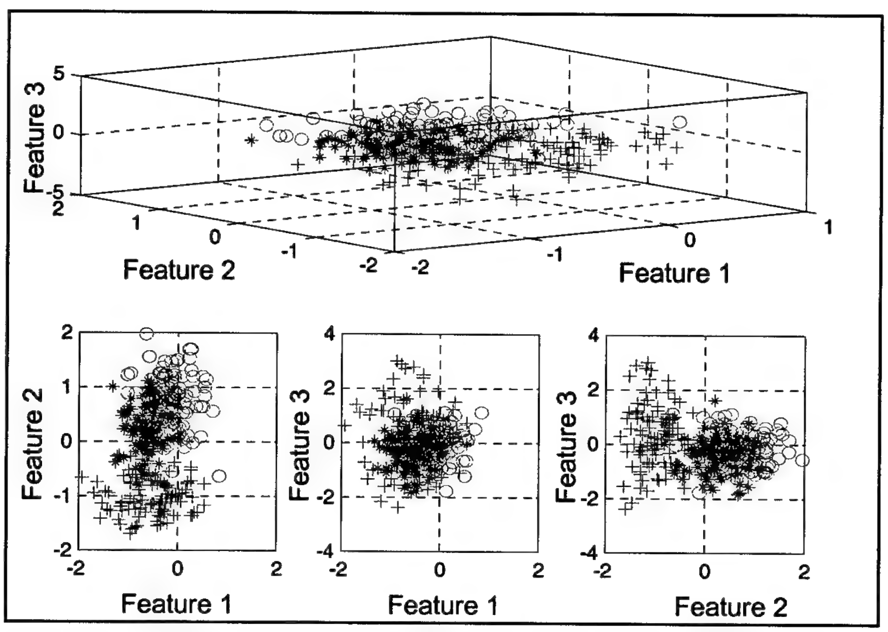


Figure IV-2. Example of 3-Feature Data for Classification (high noise).

Lastly, after creating the artificial feature vector, the data was normalized for MSNN Mod 1 implementation (as specified by Equation 3.28) and the training data covariance matrix was calculated for use by the statistical classifier. The results obtained with this parametric classifier are considered next.

B. INDIVIDUAL CLASSIFIER PERFORMANCE

1. Statistical Classifier

Chapter III defined the quadratic classifier decision rule as

$$d_i(\mathbf{x}) = \ln|\Sigma_i| + (\mathbf{x} - \boldsymbol{\mu}_i)^T \Sigma_i^{-1} (\mathbf{x} - \boldsymbol{\mu}_i) - 2\ln P_i. \tag{3.8}$$

This classifier categorized testing objects by selecting the class that resulted in the lowest value for the distance quantifier. The observations **x**, covariance matrix Σ, and mean vector μ were obtained as earlier explained. The *a priori* probability, P_i, was determined by assuming equal likelihood for all class types; $P = 1/m$, with m being the number of classes.

Recall a crucial assumption made during the derivation of Equation 3.8 required that the observations **x** form a normally distributed data set. The trials met this prerequisite by using a normally distributed random generator to produce the artificial signal features. Since these random variables were created without interdependence and are therefore uncorrelated, the joint distribution of the random variables is a product of the individual distributions. Hence, the observations are multivariate normal, indicating the quadratic classifier can be used.

Convinced that the quadratic classifier can be appropriately applied, 3000 test objects per trial were classified. For all combinations of the nine SNR levels and three input space sizes, five trials were conducted. This amounts to the classification of 405,000 test objects. For convenience, the simulation results obtained for this and all other classifiers are collected in Appendix B. Tables B-1 through B-3 contain classification confusion matrices of the statistical classifier trials and Figure B-1 plots the performance indices indicated by these tables. These results indicate that the quadratic classifier performed remarkably well under the simulated conditions. As expected,

misclassification decreased with increased SNR and feature space size. A comparison of all classification techniques will be discussed later.

2. Perceptron

The quadratic classifier models each class based on the statistical parameters of the training data. The neural network classifiers, however, use a non-parametric learning algorithm to train the network for class recognition. That is, the actual data, and not its distribution information, are used to train the network to differentiate the class.

One consequence of neuro-classifier training, however, is the absence of a unique solution in many circumstances. For instance, in the case of the perceptron neural network, different decision boundaries arise dependent on the initial weight and bias values. Recall, perceptron training was governed by the learning rules defined by Equations 3.11 and 3.12:

$$\mathbf{w}^{new} = \mathbf{w}^{old} + \mathbf{e} \bullet \mathbf{p}^{T} = \mathbf{w}^{old} + (\mathbf{t} - \mathbf{a}) \bullet \mathbf{p}^{T} \tag{3.11}$$

$$\mathbf{b}^{new} = \mathbf{b}^{old} + \mathbf{e} = \mathbf{b}^{old} + (\mathbf{t} - \mathbf{a}). \tag{3.12}$$

Since the update terms in Equations 3.11 and 3.12 are indirectly affected by the old weight and bias values through **a**, perturbations in the initial weight and bias settings can alter the final solution. In addition, there is no way to tell if an alternate weight and bias will improve network training; there is no method to determine the best starting point for perceptron training. To account for this uncertainty, the perceptron neural network was trained five times for each set of training data. For each network re-training, random generation ensured different weight and bias initializations were used. This process was then repeated with five different training data sets to test network durability. Consequently, overall the perceptron was trained twenty-five times for each noise and input space condition to provide for a more general understanding of its capabilities.

After each network training, the perceptron classified 1000 objects for each class per trial; in excess of two million objects over all simulations. Tables B-4 through B-6 and Figure B-2 summarize the results of these trials. However, not all test data was typed to one of the possible classes. As previously explained, this peculiarity arises when the

number of class possibilities (2^{μ} for a network of μ processing elements) exceeds the number of actual classes. Table IV-1 indicates the percentage of such occurrences for each SNR level and input feature size.

SNR (dB)	**3 FEATURES**	**10 FEATURES**	**50 FEATURES**
20	0.3	0.0	0.0
15	0.7	0.1	0.0
10	0.8	0.1	0.1
5	3.6	0.9	0.1
0	4.8	3.1	0.2
-5	12.5	9.9	3.0
-10	14.6	13.6	5.1
-15	17.7	16.8	8.0
-20	14.6	19.1	15.7

Table IV-1. Observed Percentage of Perceptron Non-Type Classification.

Tables B-4 through B-6 and IV-1 indicate acceptable results at positive SNR levels, but severely degraded perceptron performance with increased non-type classifications in noisy environments. In large part this is attributable to the linear decision boundaries used to separate the different classes. As SNR decreases, resulting in increased data encroachment into neighboring partitions and ultimately more cluster overlap, the perceptron's linear separators cannot adequately maintain class division. Consequently, classification performance suffered.

3. MSNN Methods

The quadratic classifier and perceptron served as benchmarks for measuring MSNN performance. For the same reason that the perceptron was subjected to multiple training cycles, each MSNN variation was trained with five different weight and bias

initializations for each set of 100 training observations per class for a three-class setup. To reiterate, the MSNN alternatives were

1. Standard MSNN
2. MSNN Mod 1: MSNN with feature space preconditioning
3. MSNN Mod 2: MSNN with projection space normalization
4. MSNN Mod 3: Standard MSNN with VMR termination

For the modifications that utilized the VMR termination parameter (variations 3 and 4), ΔV was based on 0.5% of the observations residing in the fringes of the data distribution and the VMR threshold was set at 0.90. With these stringent criteria, minimal data overlap is expected when network training secures on VMR. Unfortunately, a post-simulation record review revealed that this was not the case as network training often terminated on maximum epoch limit.

Once trained, the tuned networks classified 3000 test objects per run. As previously stated, this training/testing scheme was repeated with five different data sets to quantify network robustness. Simulation results are presented on Tables B-7 through B-9 and on Figure B-3 for the standard MSNN; on Tables B-10 through B-12 and Figure B-4 for MSNN Mod 1; on Tables B-13 through B-15 and Figure B-5 for MSNN Mod 2; and on Tables B-16 through B-18 and Figure B-6 for MSNN Mod 3. Not surprisingly, neural network performance deteriorated with increased noise levels and decreased feature space size.

In addition to these results, it is also instructive to note some characteristics of the MSNN implementation not pertinent to either the statistical classifier or perceptron neural network. For instance, plotting the surface of the mean-difference parameter, MD, over a range of weight and bias values provides insight into the behavior of the network training trajectory. Unfortunately, plotting limitations prevent graphical representations of the MD projection index and every elements of the simulated feature space since this would require hyperspace imaging. At most only two degrees of freedom could be used

to form the three-dimensional image of a particular projection index surface. Therefore, a one-dimensional classification problem was analyzed.

Figures IV-3 and IV-4 illustrate a one-dimensional classification problem and the neuron map for its sole standard MSNN processing element. In particular, Figure IV-4 confirms successful network training, as the test points for each class map to the same unique specifier and provide for maximum mean separation.

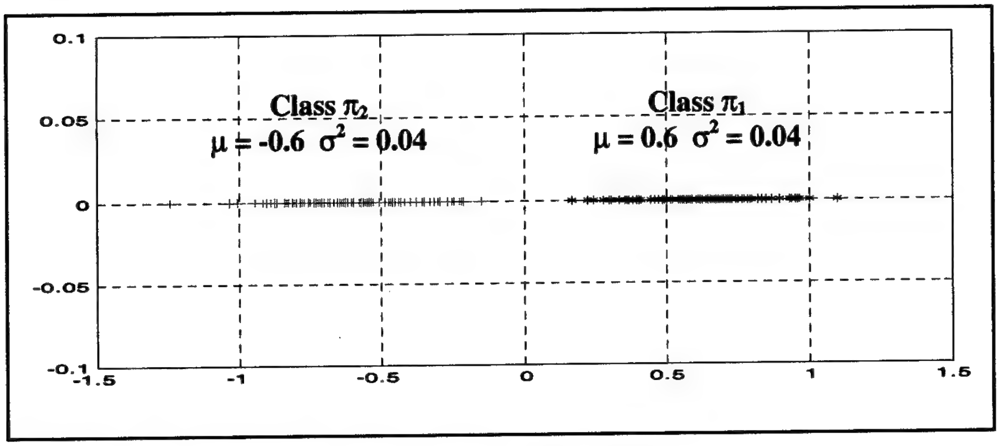


Figure IV-3. Example of 1-Feature Data for Classification.

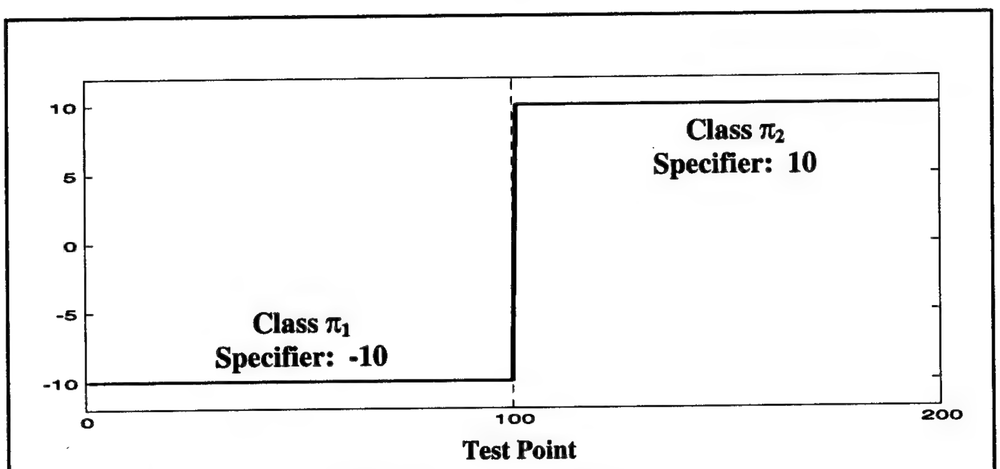


Figure IV-4. MSNN Neuron Map of 1-Feature Data.

Since the feature space is comprised of only one element, plotting the projection index surface can be achieved by considering a scalar weight and bias. This is shown in Figure IV-5. Here the upper two graphs display the MD surface characteristics in the

vicinity of the trained solution and representative contours; the lower two, a more global depiction over a wider range of weight and bias values.

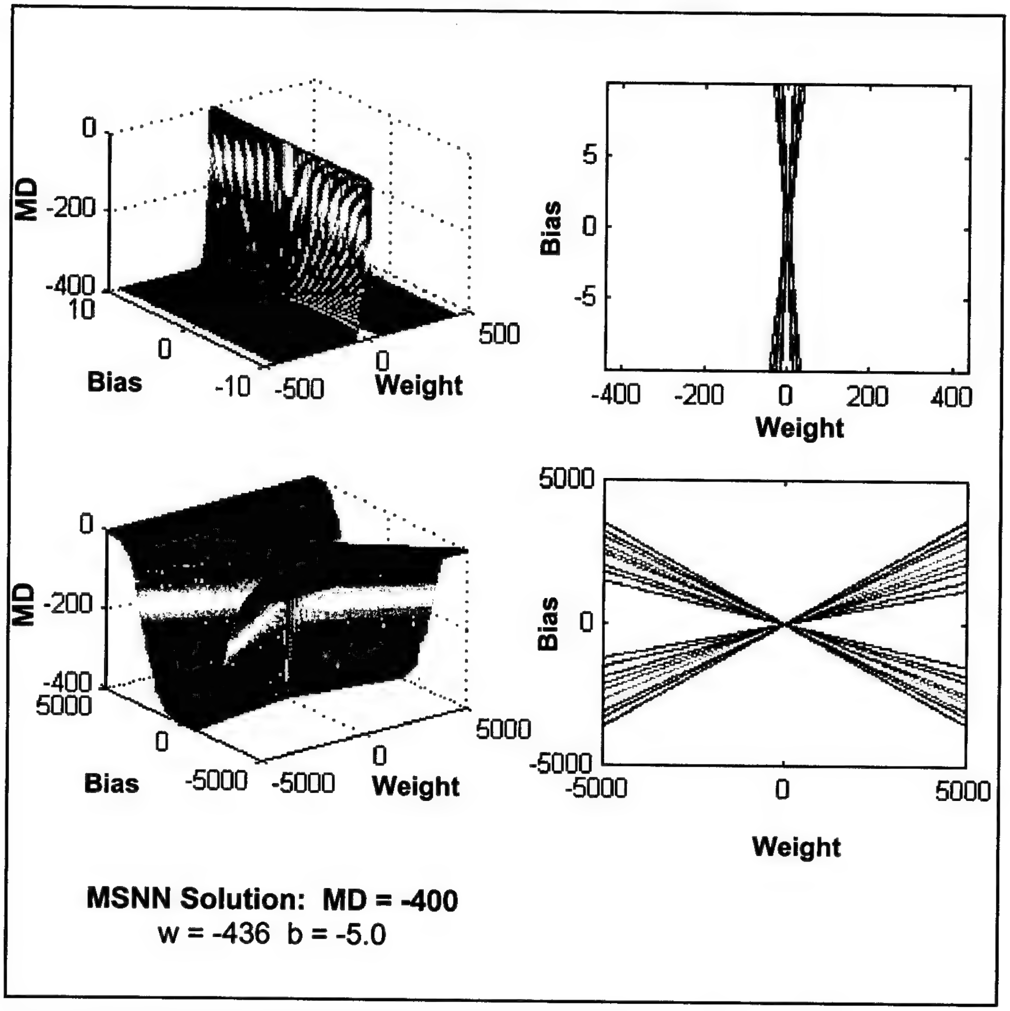


Figure IV-5. MSNN Local and Global Surface and Contour Plots.

The MSNN solution and corresponding mean-difference rating of -400 confirm the successful network training suggested by the network's neuron map. In addition, the regularity of the MD surface implies that network resolution to the final weight and bias values was unencumbered by any local minima obstacles.

Recall that a mean-difference of zero is the least desired case. Figure IV-5 shows this occurring for a weight of zero regardless of bias, and for large magnitude weight and bias values. This latter case corresponds to processing element saturation. Interestingly,

Figure IV-5 also suggests that in this trial the bias was not a vital contributor to obtaining the optimal MD value. Both the local and global plots reveal that a MD value of -400 can be attained with a relatively small bias. This, however, is primarily a function of the class data and not a general trait of mean separator transformation (Equation 3.14). In all one-dimensional cases examined, the class means were bipolar. That is, the means of the data distributions were created such that they had opposite sign. Consequently, the inherent data distribution bias (i.e., combined mean of the two classes) was near zero, indicating little need to impose an external bias to maximize mean separation.

Yet, in general, examination of the mean separator transformation suggests that the role of the bias is as a linear translator of the activation function output. The bias merely shifts the characteristic *logsig* plot horizontally. Consequently, bias can be disregarded and in its place, a second weight component considered. By considering this second weight feature, greater insight into the presence or absence of local minima and subsequently their effect on neural network performance can possibly be gained. Figures IV-6 (low noise) and IV-7 (high noise) illustrate such a two-dimensional problem. The neuron maps (Figures IV-8 through IV-22, even) and mean-difference surface and contour plots (Figures IV-9 through IV-23, odd) for the four MSNN variants follow. From these figures, it is worth noting the consistency (or lack thereof) in the neuron maps and any eccentricity in the shape of the surface plots.

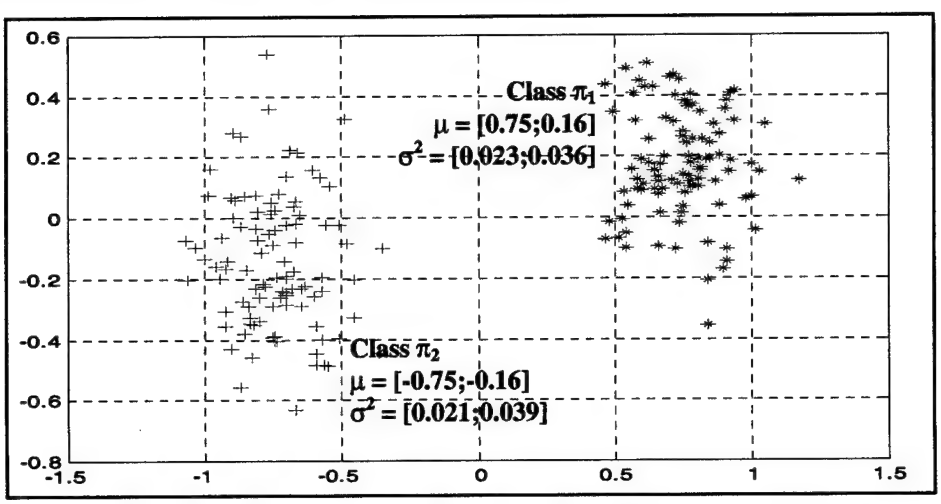


Figure IV-6. Example of 2-Feature Data for Classification (low noise).

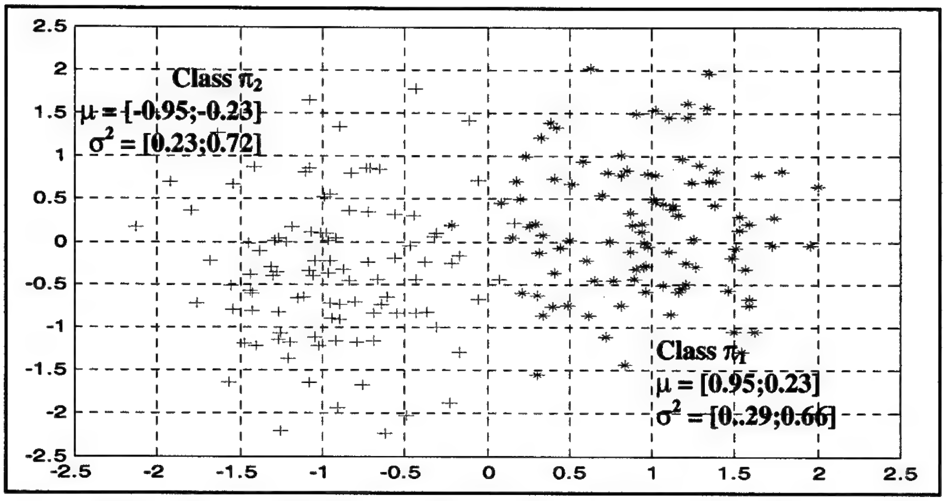


Figure IV-7. Example of 2-Feature Data for Classification (high noise).

For instance, Figures IV-10 and IV-18 suggest the futility of data preconditioning prior to network training and classification. MSNN Mod 1 consistently produced the least consistent neuron mappings and often the smallest mean spread. Further confirmed by low mean-difference indices of -134 and -174 respectively shown on Figures IV-11 and IV-19, the resulting sub-optimal mean separation led to poor classification performance.

On the other hand, the neuron maps and surface/contour plots for the remaining three MSNN variants indicate optimal network training achieved with the high SNR condition. Figures IV-8, IV-12, and IV-14 depict the maximal separation between class means and Figures IV-9, IV-13, and IV-15 report the optimal value for the mean-difference projection index. For the standard MSNN and MSNN Mod 3, this MD value is given by Equation 3.15; for MSNN Mod 2, MD_2 is calculated using Equation 3.29. Moreover, the MSNN Mod 2 mean-difference value of -10^{10} implies a sum of projection space variances much less than 10^{-7}, suggesting that transformation into the decision domain resulted in a high degree of precision and essentially no data overlap. Graphically, this accounts for the vertical slope found on the performance surface of Figure IV-13, as opposed to the more gradual descents seen on other plots. Such a favorable mapping greatly simplifies the classification task.

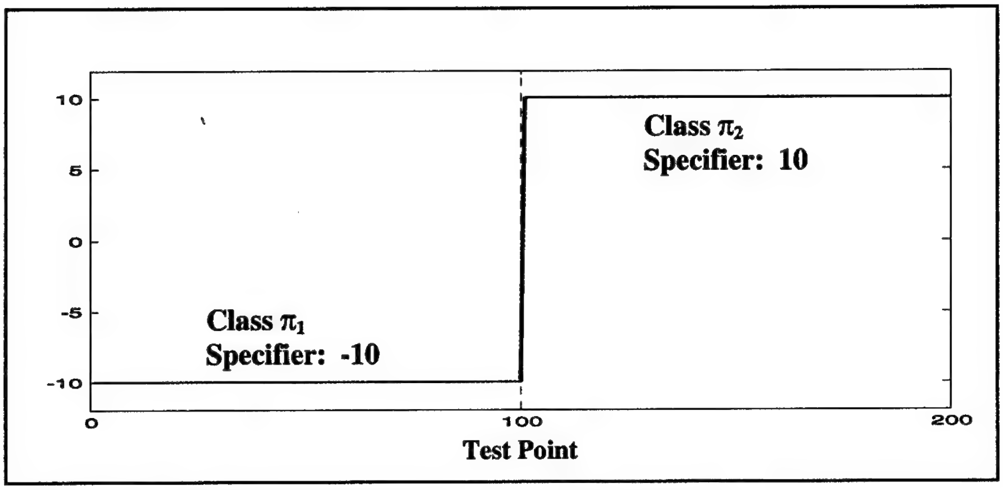


Figure IV-8. MSNN Neuron Map of 2-Feature Data (low noise).

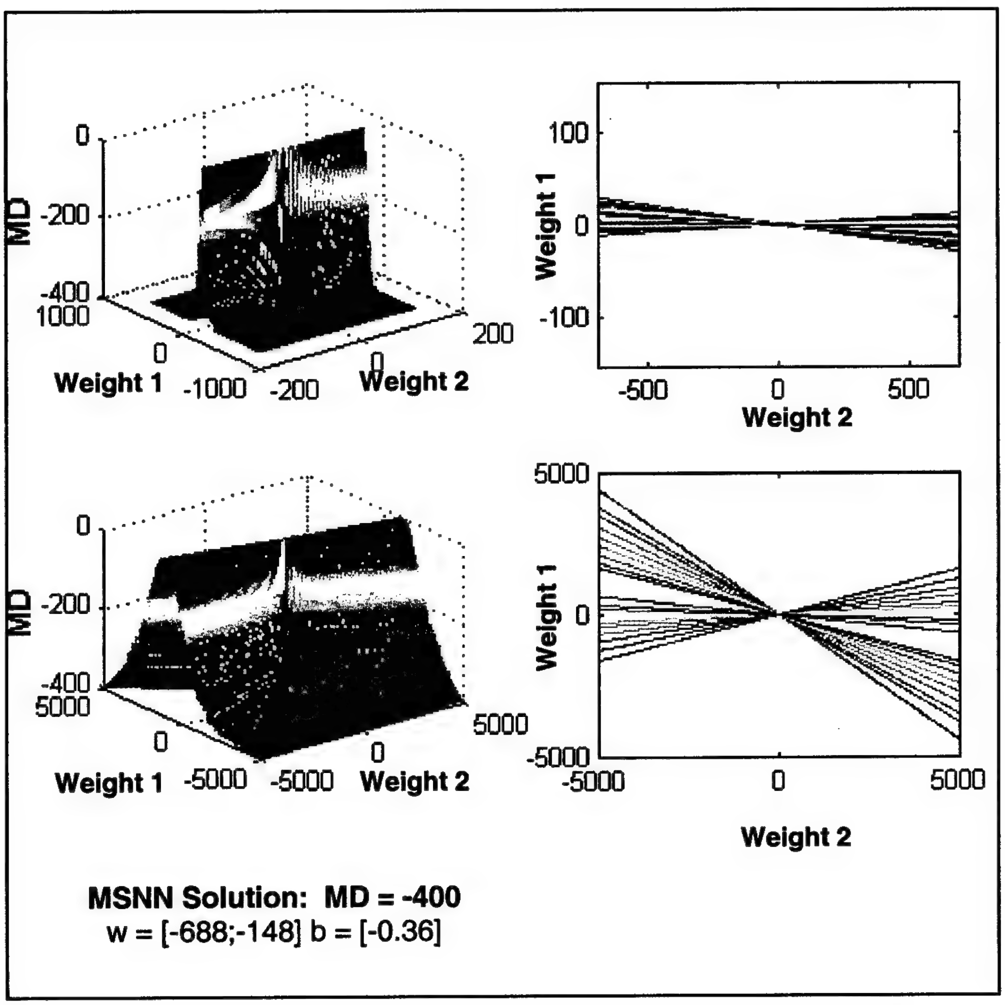


Figure IV-9. MSNN Local and Global Surface and Contour Plots (low noise).

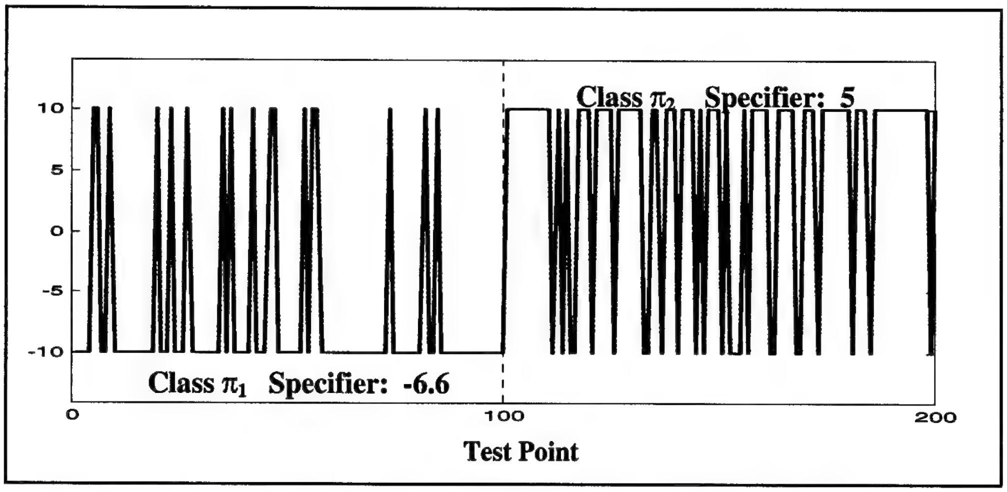


Figure IV-10. MSNN Mod 1 Neuron Map of 2-Feature Data (low noise).

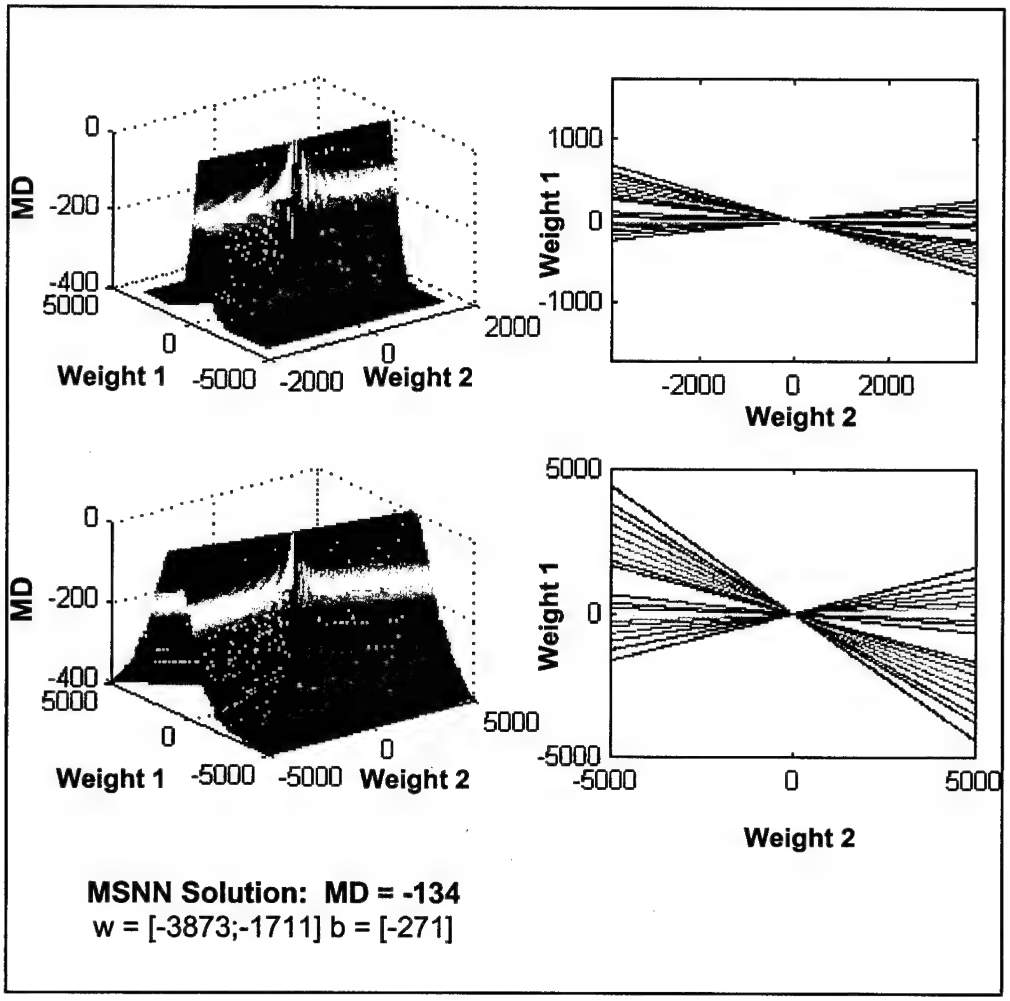


Figure IV-11. MSNN Mod 1 Local and Global Surface and Contour Plots (low noise).

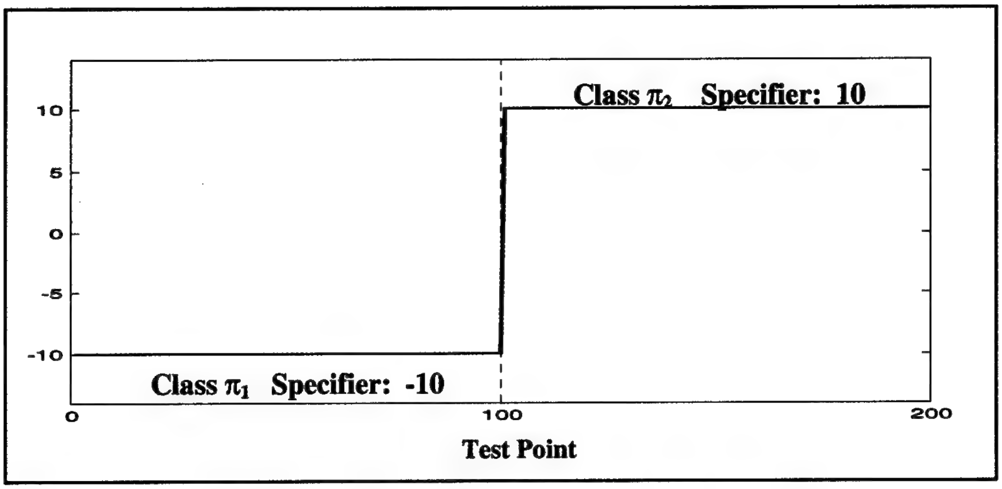


Figure IV-12. MSNN Mod 2 Neuron Map of 2-Feature Data (low noise).

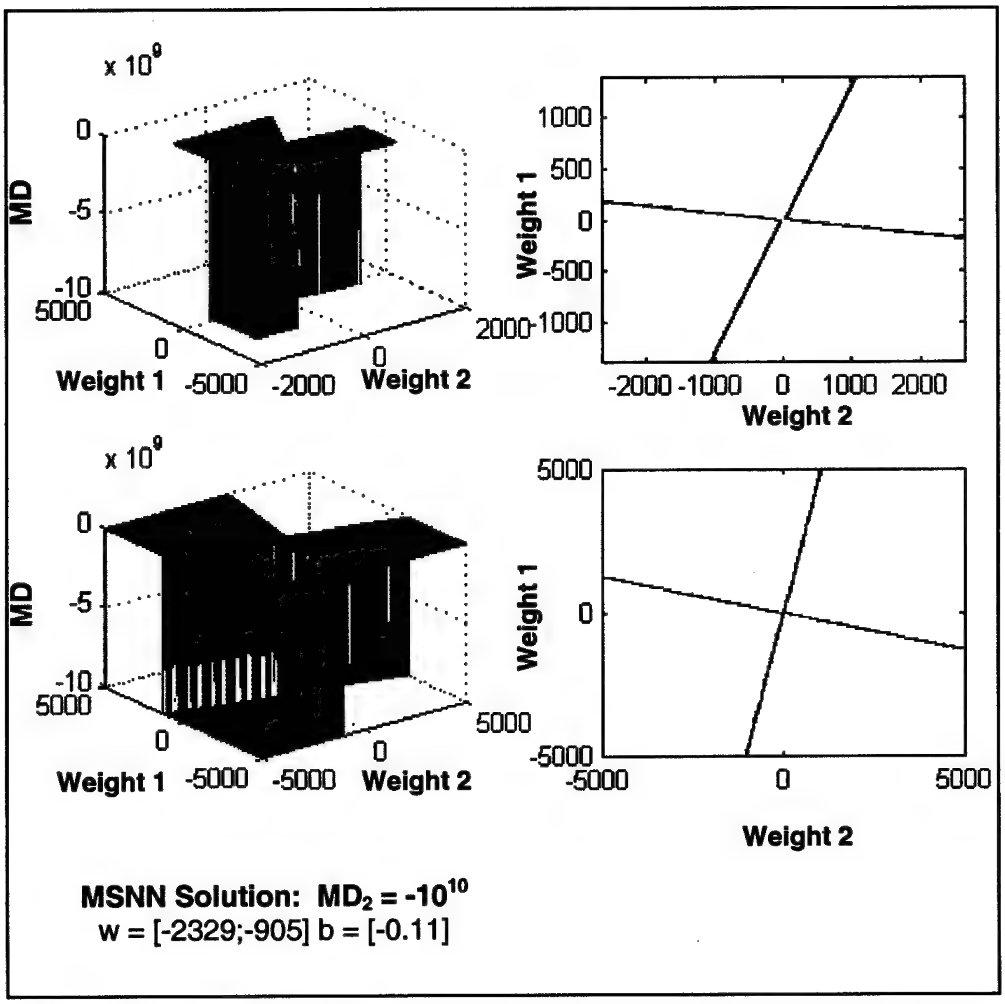


Figure IV-13. MSNN Mod 2 Local and Global Surface and Contour Plots (low noise).

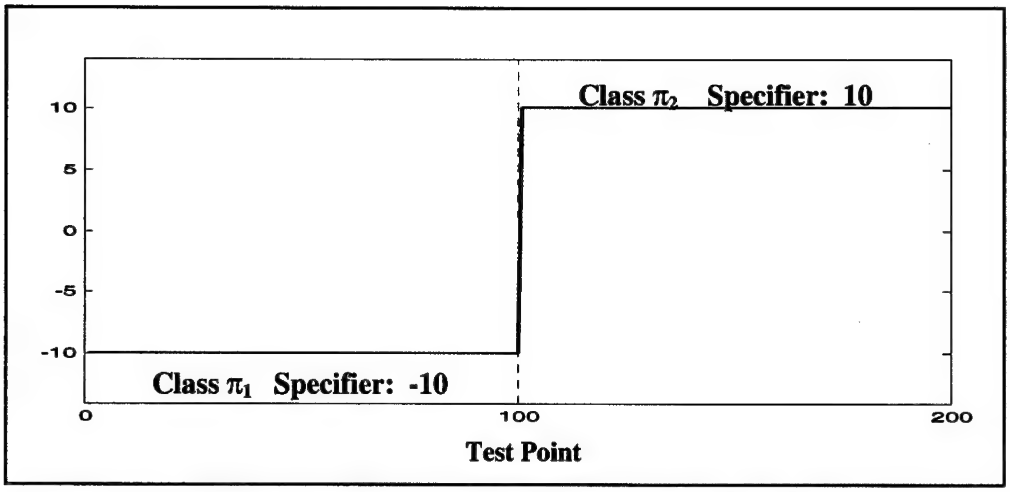


Figure IV-14. MSNN Mod 3 Neuron Map of 2-Feature Data (low noise).

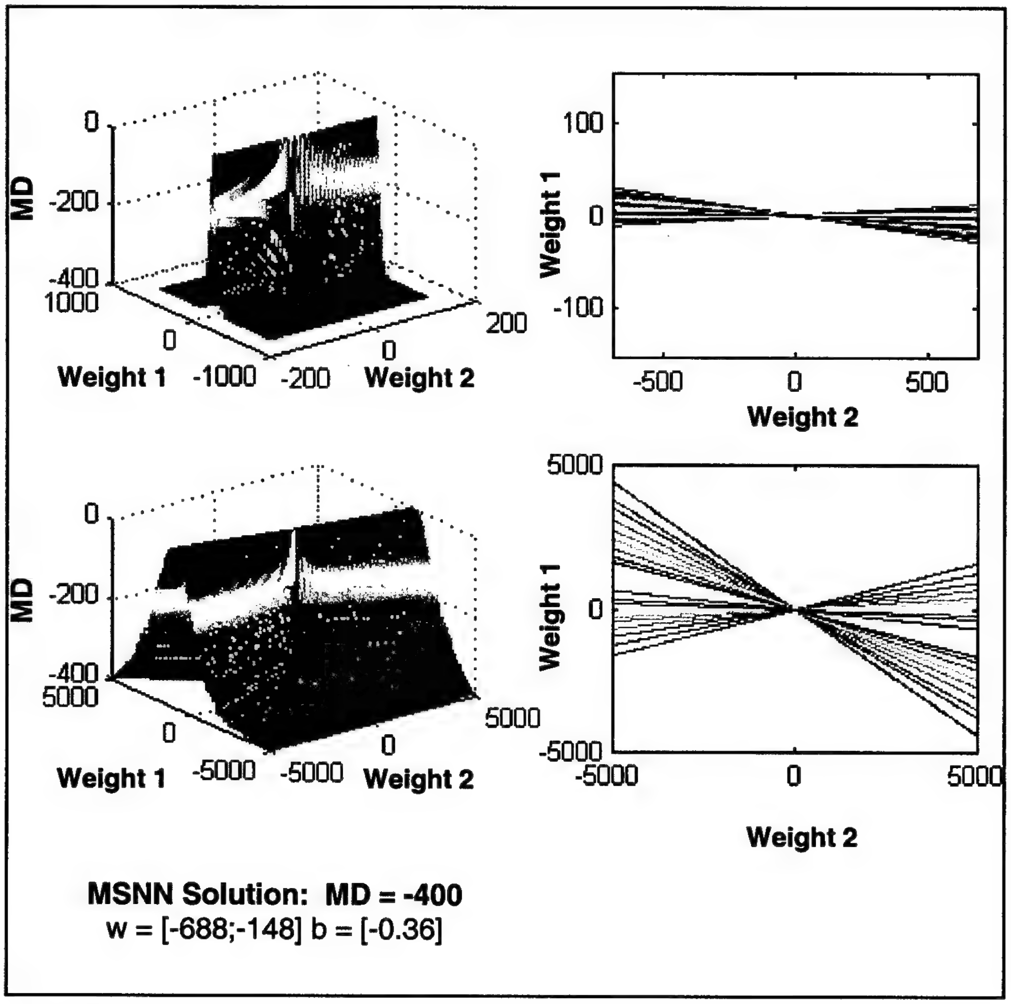


Figure IV-15. MSNN Mod 3 Local and Global Surface and Contour Plots (low noise).

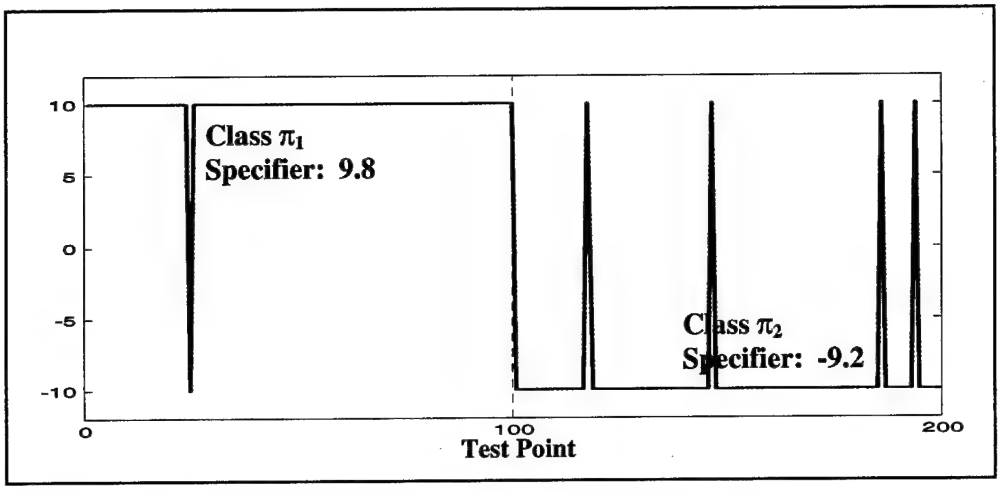


Figure IV-16. MSNN Neuron Map of 2-Feature Data (high noise).

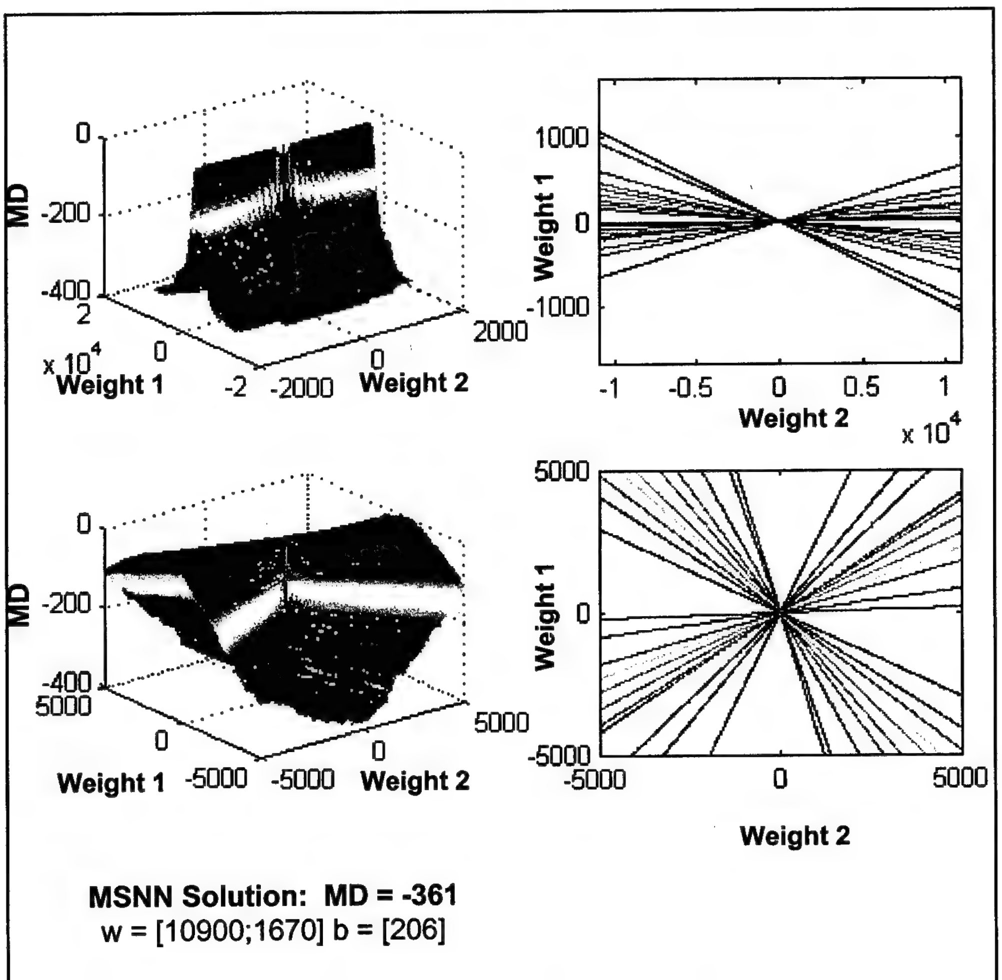


Figure IV-17. MSNN Local and Global Surface and Contour Plots (high noise).

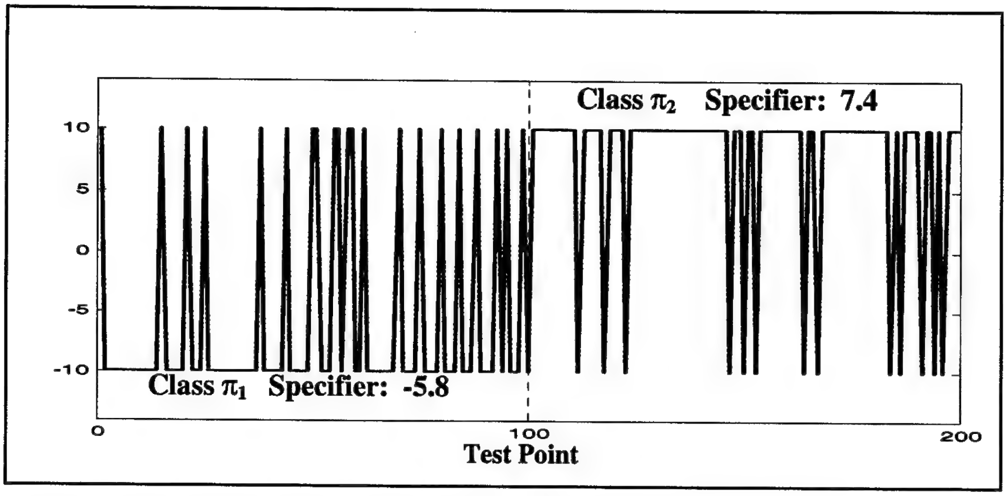


Figure IV-18. MSNN Mod 1 Neuron Map of 2-Feature Data (high noise).

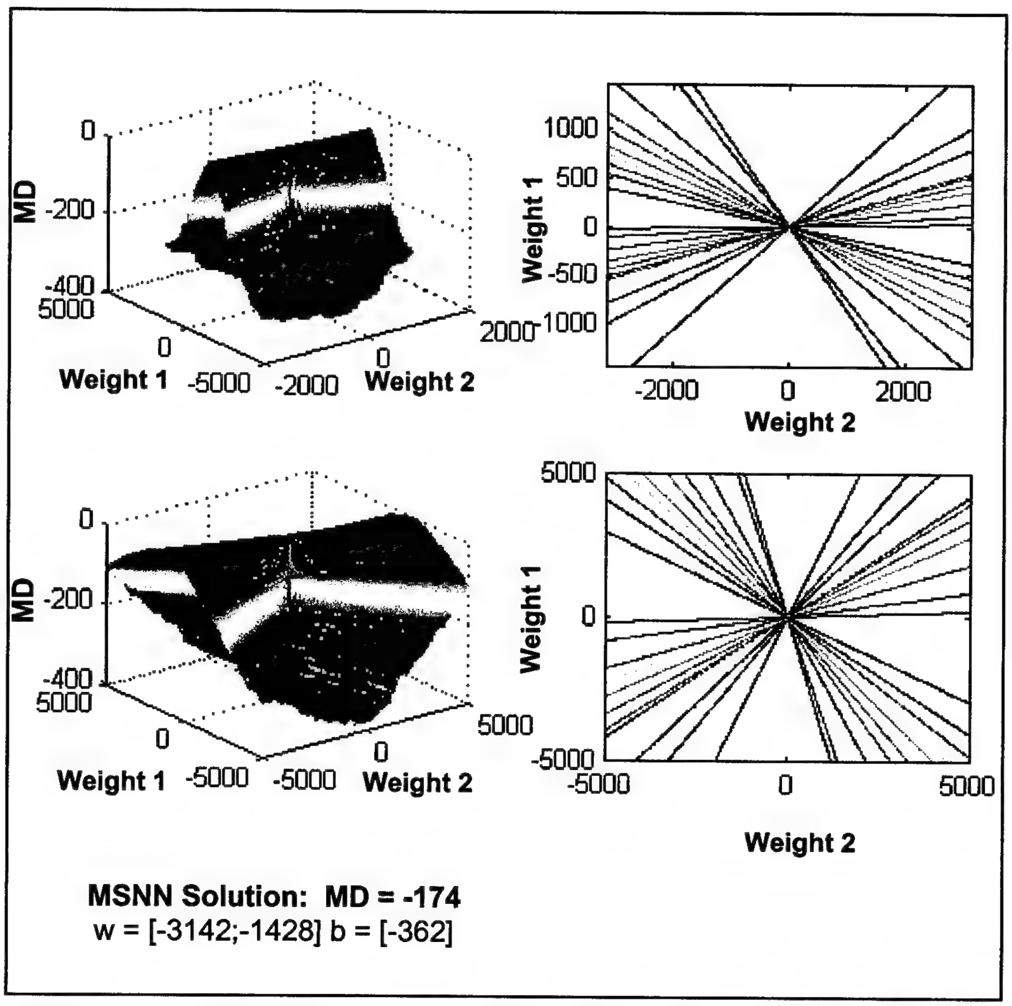


Figure IV-19. MSNN Mod 1 Local and Global Surface and Contour Plots (high noise).

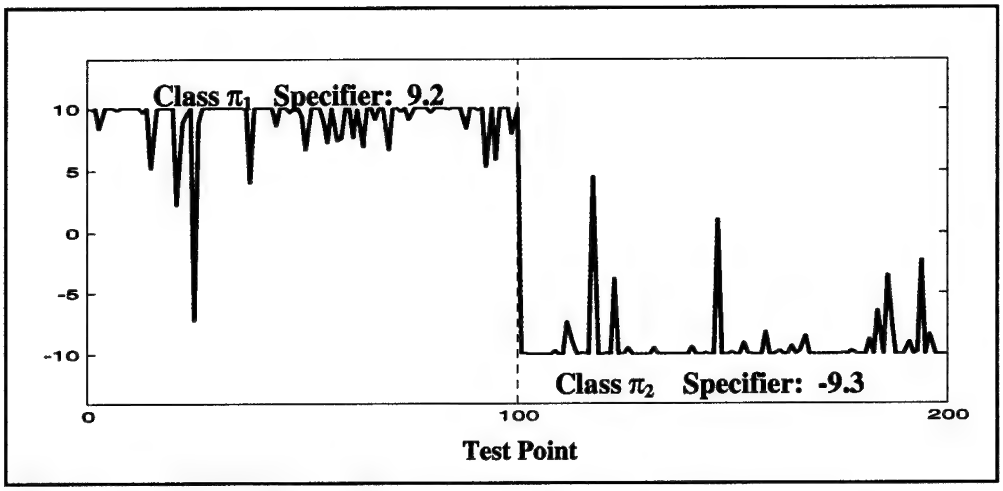


Figure IV-20. MSNN Mod 2 Neuron Map of 2-Feature Data (high noise).

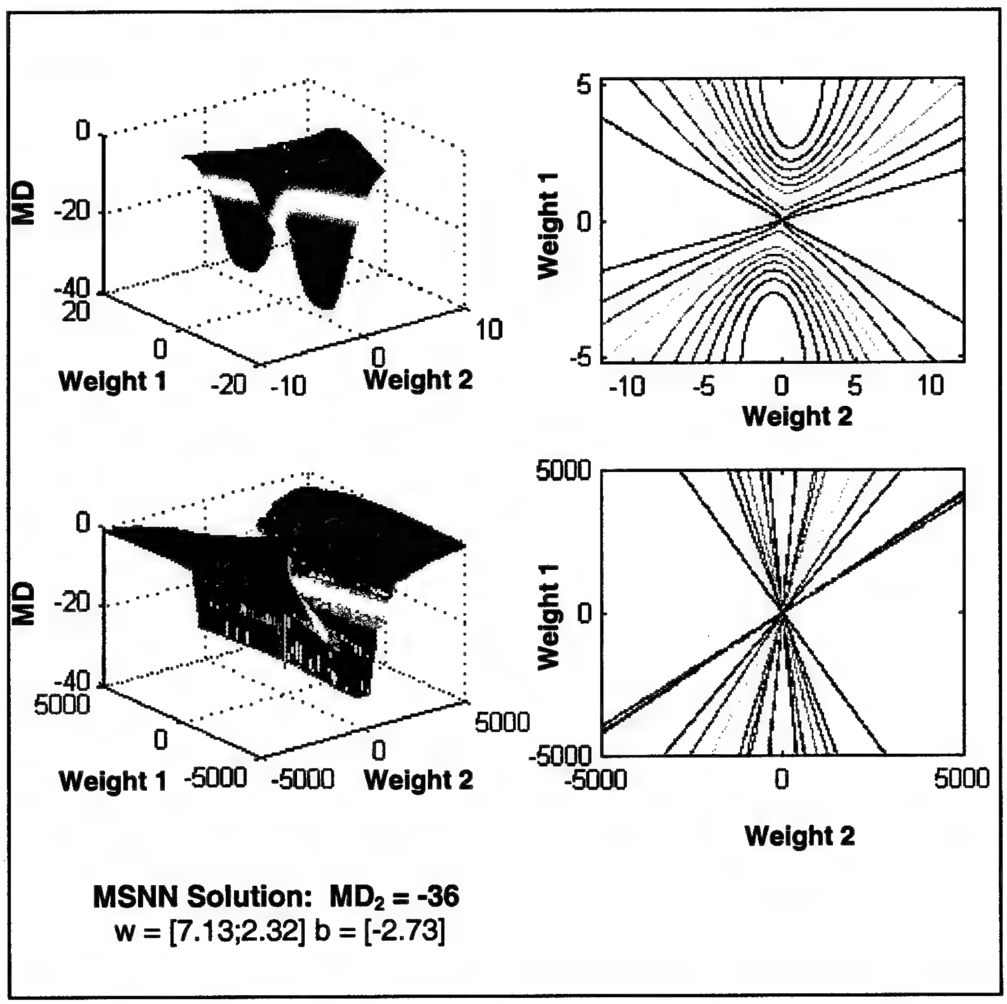


Figure IV-21. MSNN Mod 2 Local and Global Surface and Contour Plots (high noise).

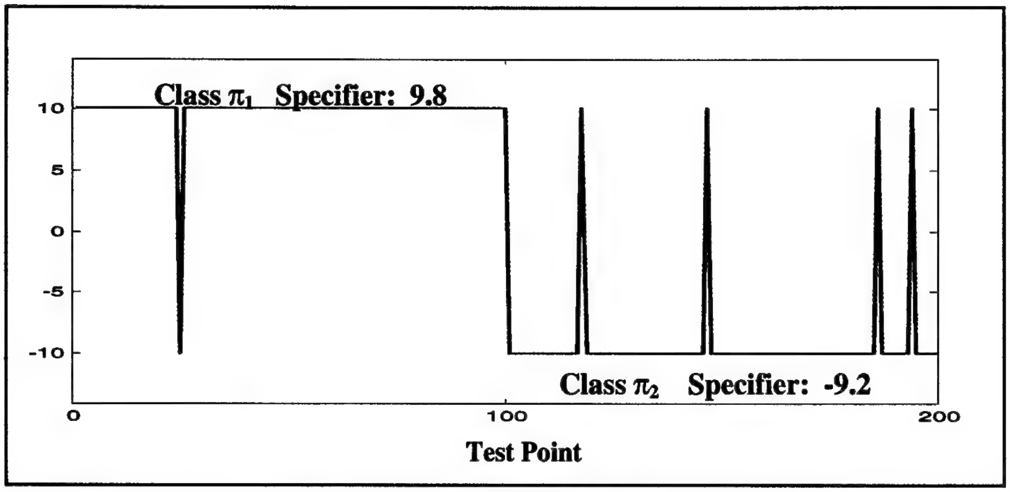


Figure IV-22. MSNN Mod 3 Neuron Map of 2-Feature Data (high noise).

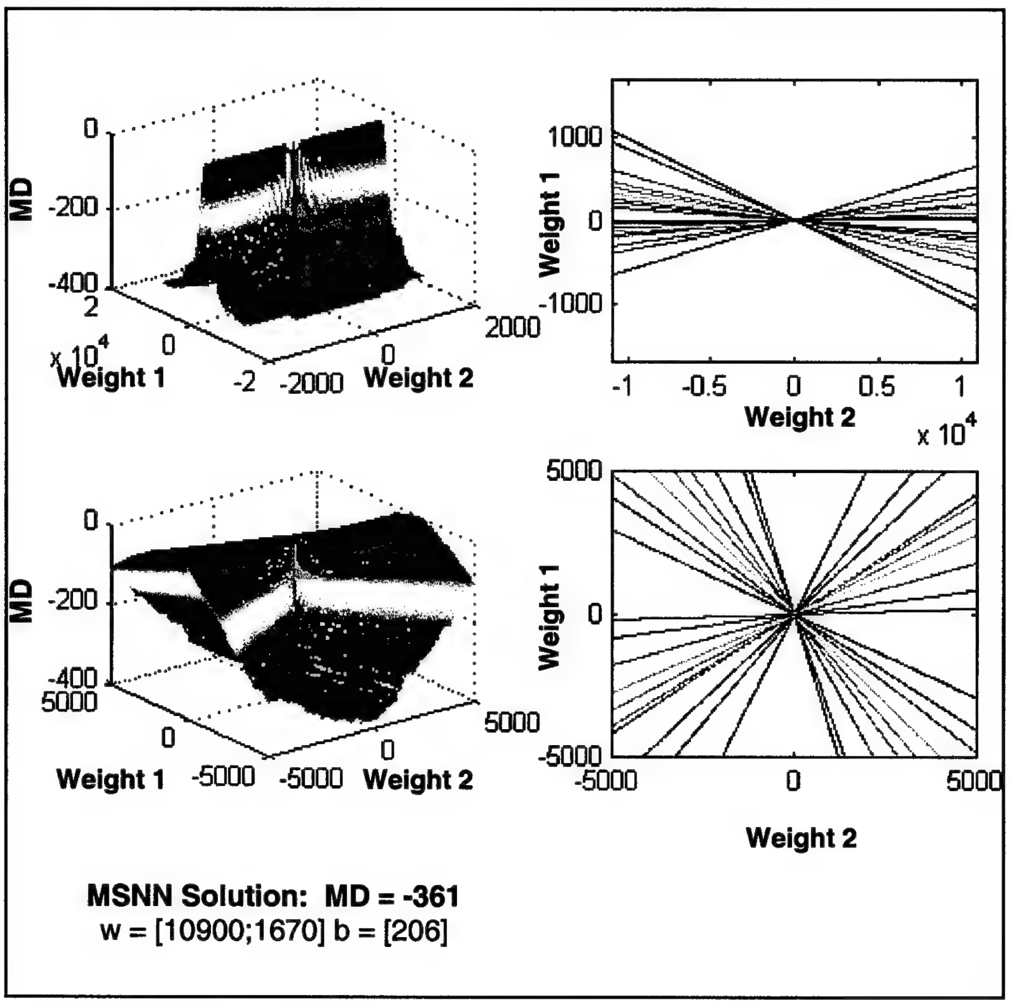


Figure IV-23. MSNN Mod 3Local and Global Surface and Contour Plots (high noise).

The superior performance of these MSNN variants relative to the MSNN Mod 1 approach is also displayed on the figures representative of high noise conditions. Moreover, these plots illustrate the effect of added noise. The wide range global plots indicate that by increasing the noise level, the area of optimal mean-difference decreases. For instance, consider the results of MSNN Mod 2 shown on Figures IV-13 and IV-21. Whereas the optimal region envelops a large area in the low noise case; with increased noise corruption, maximal MD_2 can only be attained through a narrow selection of weight values. Since fewer weight combinations will result in the optimal MD_2 value, the likelihood of attaining an acceptably trained network is lower. Consequently, more misclassifications are probable.

Also notice that the low SNR plots indicate a greater directionality towards a particular weight component, reminiscent of what was observed in the one-dimensional case. But, unlike the earlier observation, this is not a result of the simulation protocol (i.e., creating intrinsically low bias conditions). For the two-dimensional case, this directionality results from the inner product of the weight vector and actual data used, and therefore will change from simulation to simulation.

Curiously, the results obtained with the MSNN Mod 3 were exactly the same as those achieved by the standard MSNN. Recall the principle advantage of using the VMR termination criteria is that this parameter places a requirement on projection data variance in addition to projection mean spread. By considering both parameters, data overlap is minimized. Unfortunately, network training often did not secure on reaching the VMR threshold. Instead, the MSNN Mod 3 variant terminated the training phase when the number of training epochs exceeded the established limit. Because of this, future MSNN studies should increase the epoch limit and reformulate the network guidance (i.e., the learning rate rules) to take advantage of the VMR criterion while still allowing for a dynamic learning capability.

Analysis thus far has focused on the performance of the individual classification methods. The next section compares the six classification tools.

C. CLASSIFIER COMPARISON

Analysis of the classification techniques provided initial insight into their capabilities. The most revealing fact learned, however, does not concern the benefits gained by a specific method, but instead speaks to the ineffectiveness of one under the prescribed test conditions. The inability of MSNN Mod 1 (preconditioned input data) to satisfactorily classify data objects was most notable on neuron mapping plots of the input observations into the decision space (Figures IV-10 and IV-18). These figures showed imprecise projection of the input data.

The results of each classifier must be compared to determine if the neural network modification improved classification performance. Unfortunately, Figures IV-8 through IV-23 and Appendix B do not facilitate performance comparison of the six classification techniques. This contrast, however, can be gleaned by fusing the information found on Figures B-1 through B-6 into three plots differentiated by input space size, shown as Figures IV-24 through IV-26. For the purposes of this evaluation, reliable classification capabilities are demonstrated at each SNR level if the average correct classification percentage exceeds ninety-percent.

Using this standard, the statistical classifier achieved the most accurate level of performance. For a small feature space, the parametric classifier attained over ninety-percent accuracy at a SNR of 7 dB. As input space dimensionality increased to fifty features, this performance level was maintained for all SNRs. This high classification success can be attributed to the classifier's ability to minimize classification error, as alluded to in Chapter III. Since the artificial features were normally distributed and independently created, the data set was well conditioned, allowing for optimum performance of the statistical classifier.

For the MSNN variants, Figures IV-24 through IV-26 do not clearly indicate which technique performs best. The greatest distinction is discernable in the three-feature input space. As shown on Figure IV-24, there is little difference between the performance of the standard MSNN and MSNN Mod 2, with each maintaining the ninety-percent accuracy level down to 5 and 6 dB, respectively. MSNN Mod 3 met this

limit at 11 dB and then paralleled the standard MSNN and MSNN Mod 2 algorithms with a slight offset. Not unexpectedly, MSNN Mod 1 proved to be the least successful technique, with all SNRs resulting in sub-ninety-percent accuracy.

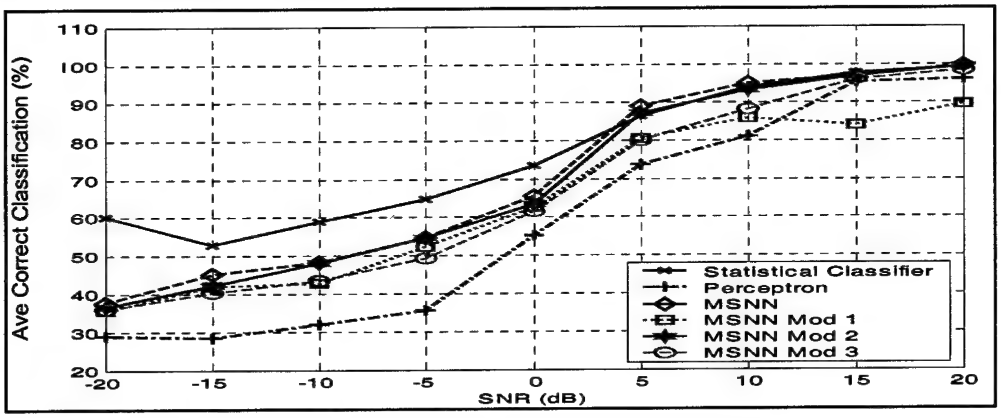


Figure IV-24. Performance Comparison: Simulated Features (3).

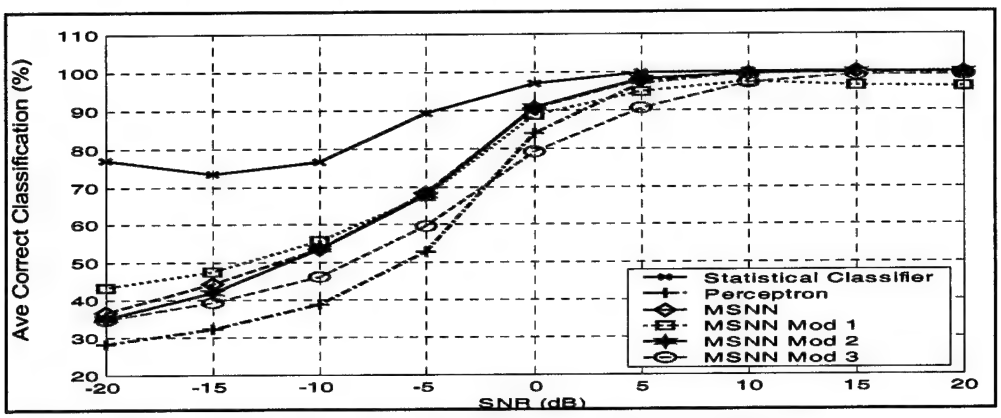


Figure IV-25. Performance Comparison: Simulated Features (10).

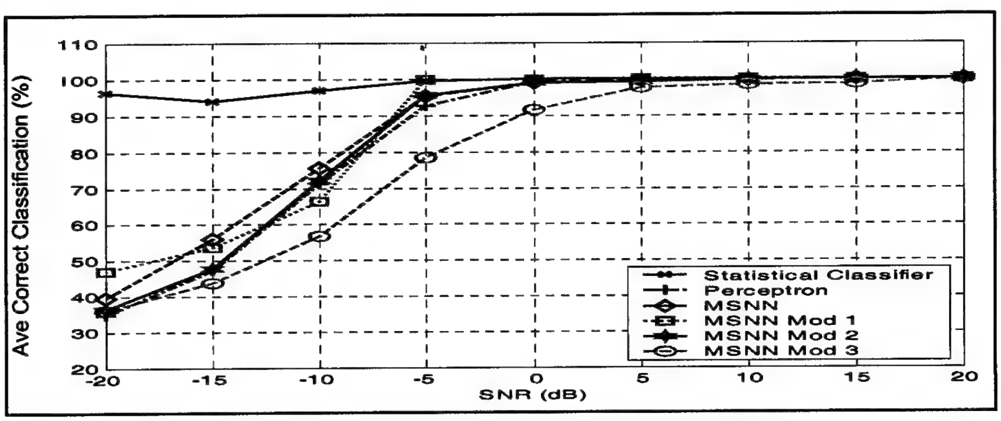


Figure IV-26. Performance Comparison: Simulated Features (50).

In general, as the number of input features increased, all classifiers showed greater classification success. Moreover, MSNN Mod 1 surprisingly showed improved performance equal to the standard MSNN and MSNN Mod 2 methods in the ten- and fifty-dimension feature spaces. With these feature space dimensionalities, ninety-percent classification accuracy was sustained down to 0 dB and –7 dB, respectively, for the three MSNN variants listed.

Curiously, the MSNN Mod 3 variant demonstrated the least amount of improvement. For instance, Figure IV-26 indicates twenty-percent disparity between this hybrid method and the standard MSNN at SNRs of -5 dB and -10 dB. This difference and lack of significant improvement can again be attributed to MSNN Mod 3 terminating its training on maximum epoch limit instead of on VMR threshold. Unlike the standard MSNN that re-initializes its weights and bias and retrains the network when network learning ceases prior to satisfactorily training, MSNN Mod 3 implements the weight and bias it had attained when a termination parameter setpoint is reached. Since acceptable network training may not have been achieved, poor classification performance would results.

With an input space dimensionality of three, the perceptron performed on par with the MSNN Mod 3 variant to 15 dB. Below this SNR level, perceptron performance decline can be accredited to greater data noise; the resulting increased data overlap limiting the network's ability to establish linear class boundaries.

D. SUMMARY

Chapter IV utilized simulated data consisting of artificial feature elements to measure classification method performance. Considering varying noise and input space size, data sets of 300 training and 3000 testing objects were created. For the statistical classifier, ten such data sets were created for all combinations of SNR and feature space size. For the neural network trials, five data sets were simulated. In addition, because of a dependence on weight and bias initialization, the neural networks processed each set of observations from five different starting conditions.

Considering the empirical results compiled on Figures IV-24, the statistical classifier attained the greatest level of classification success. The standard MSNN algorithm and MSNN Mod 2 were the next most successful, followed by MSNN Mods 1 and 3. At high SNR, perceptron performance was comparable to the other classifiers; but at increased noise levels, dropped off precipitously.

Results for ten- and fifty-feature input spaces are also shown as Figures IV-25 and IV-26. Due to increased dimensionality, all classifiers performed equally well. In those instances where the performance of the different classifiers deviated, classification levels were below ninety-percent. Therefore, comparison of the methods is inconsequential since all would be considered unacceptable.

Overall, Chapter IV sought to establish classifier feasibility. Disappointingly, the trial simulations did not show a significant difference between the MSNN variants studied. The next chapter attempts to make this distinction by examining near real world application of these methods through simulation and classification of modulated communication signals.

THIS PAGE INTENTIONALLY LEFT BLANK

V. CLASSIFICATION OF MODULATED SIGNALS

The intent of this thesis is to demonstrate the robustness of the MSNN variants in classifying data to the appropriate signal class. In Chapter IV, the performance of these neuro-classifiers, as well as that of a quadratic statistical classifier and a perceptron neural network, were evaluated based on the accuracy attained in categorizing random vectors composed of artificially simulated features. In this chapter, these classification tools will be used to separate data objects consisting of features extracted from synthetic communication signals. The process of feature extraction is introduced prior to discussing the experimental procedure and simulation results. MATLAB program codes used during these trials are presented in Appendix C.

A. FEATURE EXTRACTION

By identifying the class to which a signal belongs, classification tools convert data to information, freeing the operator from the tedium of manually associating objects to class. Such processes consequently enable the military commander to garner knowledge and wisdom efficiently, thereby allowing him to more effectively interpret, predict, and appropriately respond to the environment. In short, these classification tools increase his situational awareness and improve his decision-making capability.

However, automating such capabilities is not a trivial endeavor. This thesis has identified and demonstrated tools that facilitate information and knowledge management, but has neglected to specify how in real-world applications the observation vectors would be obtained. Indeed, "a major problem in the area of modulation recognition is the choice of distinctive marks for distinguishing between the different types of modulation without knowledge of modulation parameters" (Reichert, 1992, p.221).

In trying to determine the extraction method to employ, most techniques avoid time-domain features because they have been shown to lack robustness at low SNR (Ghani and Lamontagne, 1993, p. 111). A noteworthy exception to this may be the exploitation of hidden periodicities found in cyclostationary signals. As recognized by Reichert, attributes of the complex envelope of linearly modulated signals, when mapped

to a single power spectral line by an appropriate transformation, uniquely identify the underlying modulation type. Moreover, this method is robust in noisy environments since uncorrelated noise will not add spectral lines that could be read as modulated signal. (Reichert, 1992)

In another approach, the features of interests were counts falling into subdivisions of the signal plane. Conceptually, this gives an empirical distribution of the observed data. Then using a distance metric, the Hellinger distance, this distribution can be compared to known signal densities. The signal corresponding to the lowest distance measure is chosen as the class type of the observations. (Huo and Donoho, 1998)

Despite interest in these techniques, their incompatibility with neural networks and mathematical complexity precluded implementation in this study. So instead, spectral characteristics were used.

Several studies have utilized spectral coefficients as features for classification. Duzenli used time-frequency characteristics obtained through wavelet decompositions to categorize underwater signals (Duzenli, 1998), while others used Fourier transform coefficients for analysis (Ghani and Lamontagne, 1993), (Lallo, 1999). This thesis also extracted features from the Fourier domain. The creation of these simulated signals and execution of empirical trials is discussed next.

B. SIGNAL SIMULATION

1. Signal Construction

The signal plane consisted of three communication modulation types corrupted by varying degrees of additive, white Gaussian noise. The model for constructing these signal realizations is represented by Equation 5.1 as

$$x(t) = s(t) + n(t), \tag{5.1}$$

with s(t) being the uncorrupted signal; n(t), the additive white Gaussian noise component; and x(t), the corrupted signal. Specifically, the three signal classes simulated were binary amplitude shift keying (2-ASK), binary phase shift keying (2-PSK), and binary frequency shift keying (2-FSK). The governing equations for these signal types are

$$s_{ASK}(t) = \frac{A_k}{\sqrt{T}}\sin(2\pi f_c t) \qquad \text{for } 0 \le t < T \tag{5.2}$$

$$s_{PSK}(t) = \sqrt{\frac{2}{T}}\sin(2\pi f_c t + \varphi_k) \qquad \text{for } 0 \le t < T \tag{5.3}$$

$$s_{FSK}(t) = \sqrt{\frac{2}{T}}\sin(2\pi(f_c + \Delta f_k)t) \quad \text{for } 0 \le t < T. \tag{5.4}$$

All signal types had a carrier frequency, f_c, of 40 MHz and a signal bit period, T, of 10^{-7} seconds, resulting in four cycles per message bit. Sampling the continuous signal at 500 MHz gives a discrete time representation of 12.5 samples per cycle or 50 samples per bit.

Different signal realizations were then constructed by encoding random baseband binary messages with the different modulation types. For 2-ASK, the random message determined if the signal amplitude, A_k, was zero or one. For 2-PSK, the random message determined if the phase shift, ϕ_k, was zero or π radians. For 2-FSK, the random message determined if the adjacent frequency spacing , Δf_k, was zero or 10 MHz. The normalized sum of squares over all time-domain components then furnished the signal power of each realization. Using this signal power, the noise power for the desired SNR level was determined according to

$$SNR = 10\log_{10}\left(\frac{P_s}{P_n}\right) \tag{5.5}$$

and added to the signal realization (Equation 5.1). As with the artificial feature simulations, SNRs of ±20 dB, ±15 dB, ±10 dB, ±5 dB, and 0 dB were considered, as well as a no-noise case. The final signal representation for each realization was attained by normalizing each corrupted signal by its overall power level.

To extract the features needed for classification, the time-domain signals were projected into the Fourier domain where the spectral coefficients directly relate to the signal's power spectral density. To identify the needed signal characteristics, two techniques were attempted. The more general approach identified a signal's largest spectral component and extracted those frequencies whose coefficients exceeded a certain percentage of this maximum value. Repeated for 100 training realizations of each signal

type, the common frequencies from this set of feature vectors specified the identifying attributes for each signal class. A compilation of these class characteristics provided the final feature set and dimensionality for the signal space. The training and testing data objects of each class would utilize this full description of the signal space, and not just the features initially selected for the individual class type.

Unfortunately, this method proved unreliable. Often one or two components may typify a certain class, while thirty or more may be extracted from another. Because of this disparity, the signal space did not fairly distinguish each class, especially those represented by a small number of attributes. Hence, a more rigid feature extraction scheme was considered.

Previous studies had ascertained that the information needed to discriminate different modulation types was contained within a window centered on the carrier frequency (Ghani and Lamontagne, 1993, p. 113). Using a 1000-point discrete Fourier transform and knowing the sampling frequency, the carrier frequency was found to reside at bin 80. For the 2-FSK signals, a second predominate spectral spike also appears at 50 MHz, the sum of the carrier frequency and adjacent frequency spacing; bin 100.

Knowing the bin location of the 40 MHz carrier frequency, three schemes were used to extract features from the main and first side lobes of the spectrum. In the first case, the fifty-one spectral coefficients from between bin 30 and 130 (i.e., every other frequency bin) were used as the extracted features. The second case used the coefficients of every fourth frequency; the last, every tenth bin. Respectively, the second and third schemes constitute a signal space of twenty-six and eleven input variables. Figures V-1 through V-3, verify that the selected spectral components do distinguish the three signal classes. Taken for the eleven feature signal space, these time and spectral representations of noise-free simulated communication signals specifically show that 2-ASK has more spectral energy concentrated in the carrier frequency than 2-PSK. The spike at bin 80 is larger and the side lobes are more subdued for 2-ASK. Also, these two modulation can be separated from 2-FSK by the absence of the second frequency spike at bin 100.

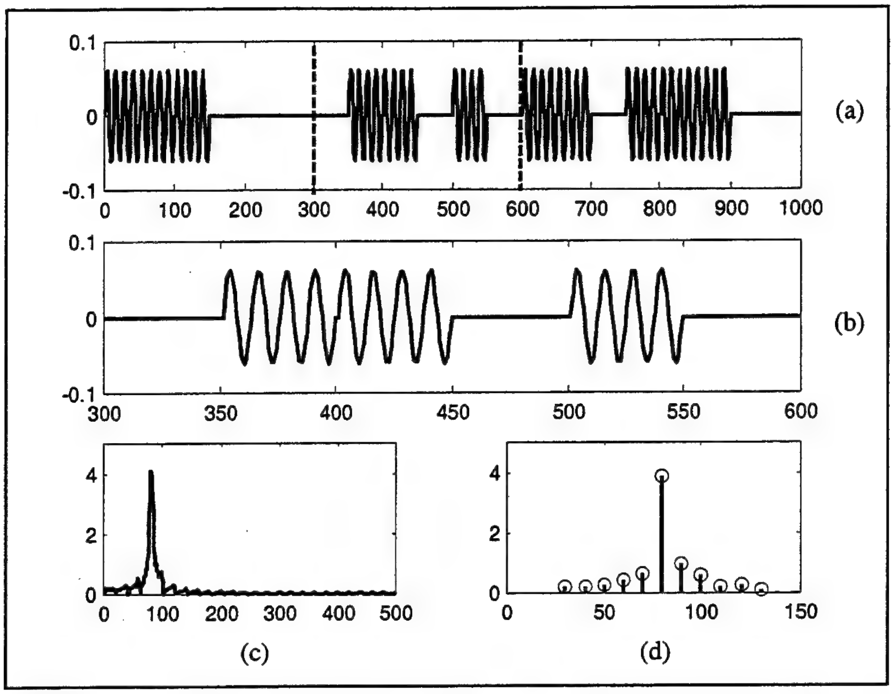


Figure V-1. Simulated 2-ASK Signal (no noise). (a) modulated signal vs sample number (b) enlargement of modulated signal vs sample number (c) spectral characteristics vs frequency bin (d) extracted frequency bins.

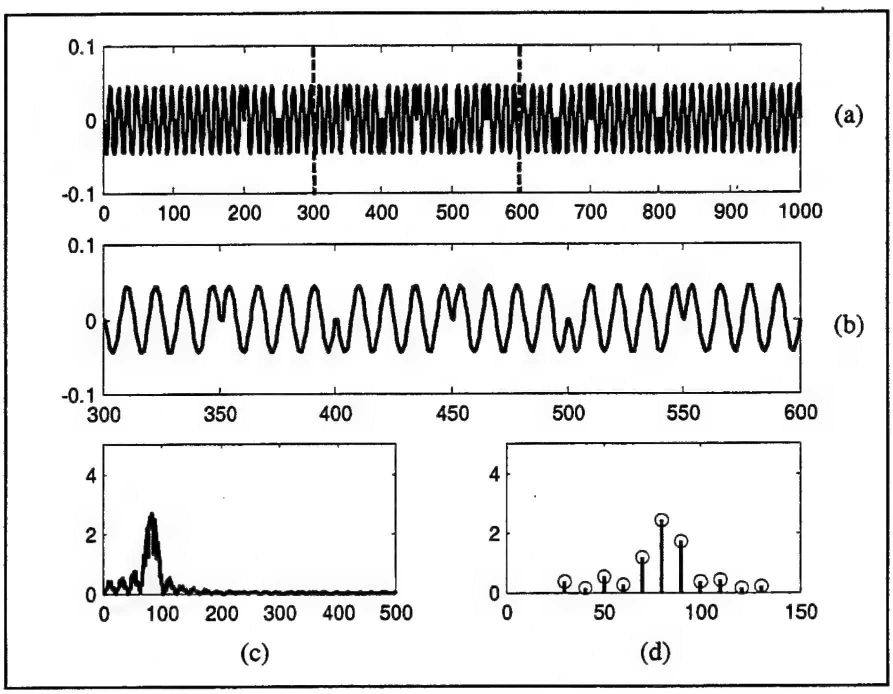


Figure V-2. Simulated 2-PSK Signal (no noise). (a) modulated signal vs sample number (b) enlargement of modulated signal vs sample number (c) spectral characteristics vs frequency bin (d) extracted frequency bins.

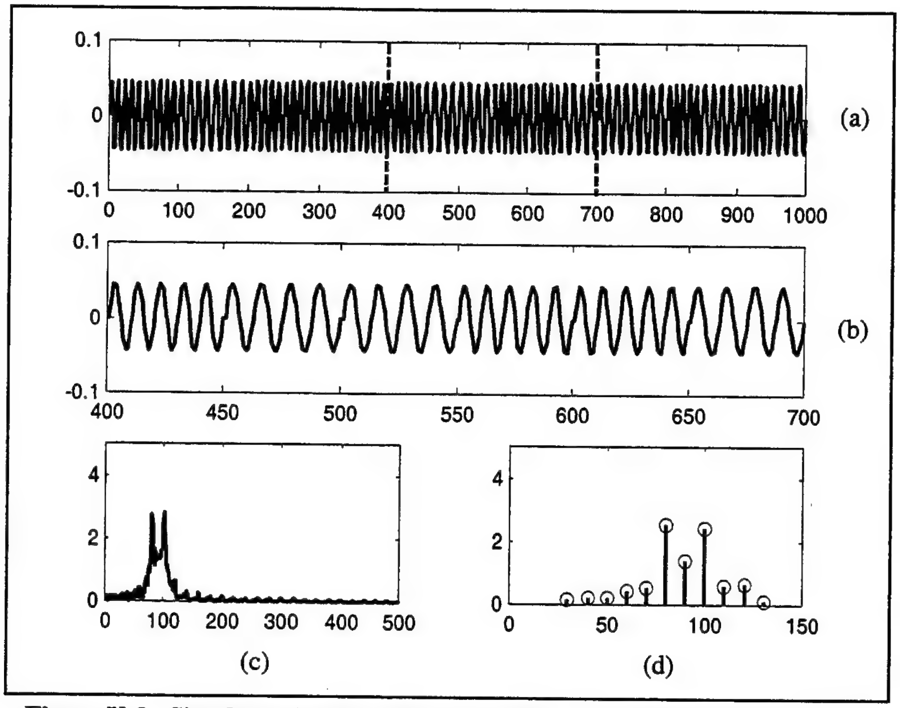


Figure V-3. Simulated 2-FSK Signal (no noise). (a) modulated signal vs sample number (b) enlargement of modulated signal vs sample number (c) spectral characteristics vs frequency bin (d) extracted frequency bins.

Examples of noise-corrupted signals are shown on Figures V-4 through V-6 for a SNR of 20 dB, and on Figures V-7 through V-10 for an SNR of 10 dB. In these figures, plot (a) depicts a sample of the uncorrupted normalized time-domain signal versus sample number; plot (b), the noise-corrupted version versus sample number. Plot (c) shows the spectral characteristic of the corrupted signal as a function of frequency bin, while plot (d) displays the frequency bins chosen for an eleven-feature input space.

In retrospect, however, the chosen frequencies should have been more judiciously selected, such as through a principal component analysis or other feature reduction method that more compactly describes the signal space (Duzenli, 1998), (Duzenli and Fargues, 1998), (Fargues and Duzenli, 1998), (Brunzell and Eriksson, 1999). Not having done so led to inconclusive results for classification of noise-corrupted signals.

Lastly, recognize that a rudimentary communication signal model corrupted by only additive, white Gaussian noise was considered. More complex modulation schemes, multi-path receptions, intersymbol interference, interlaced signals, and different fading

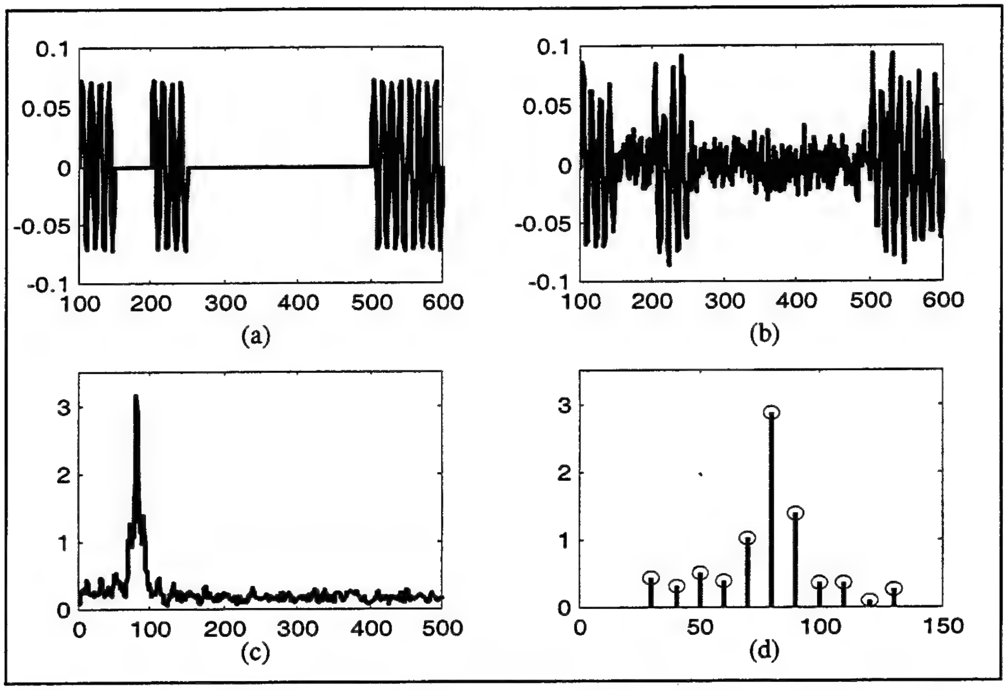


Figure V-4. 2-ASK Signal. (a) enlargement of modulated signal vs sample number (b) enlargement of corrupted signal vs sample number (SNR = 20 dB) (c) spectral characteristics vs frequency bin (d) extracted frequency bins.

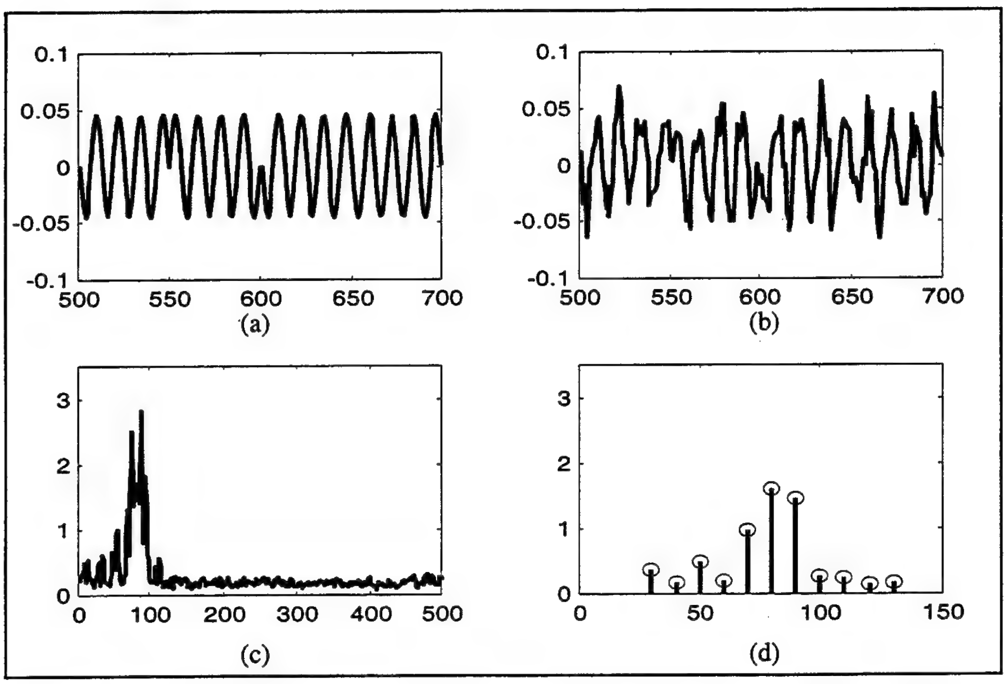


Figure V-5. 2-PSK Signal. (a) enlargement of modulated signal vs sample number (b) enlargement of corrupted signal vs sample number (SNR = 20 dB) (c) spectral characteristics vs frequency bin (d) extracted frequency bins.

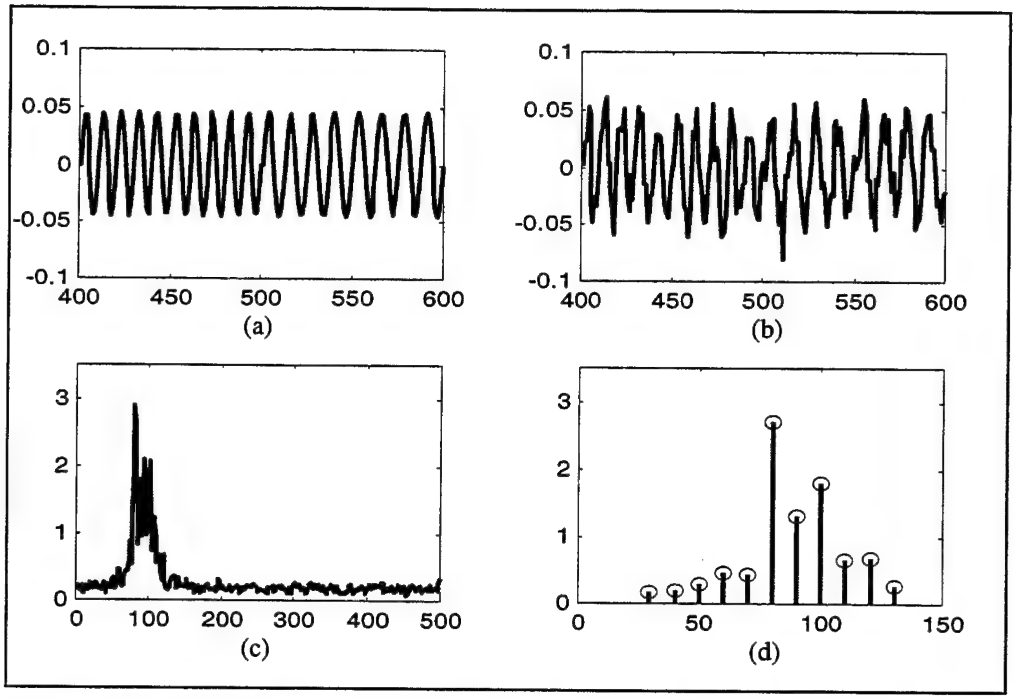


Figure V-6. 2-FSK Signal. (a) enlargement of modulated signal vs sample number (b) enlargement of corrupted signal vs sample number (SNR = 20 dB) (c) spectral characteristics vs frequency bin (d) extracted frequency bins.

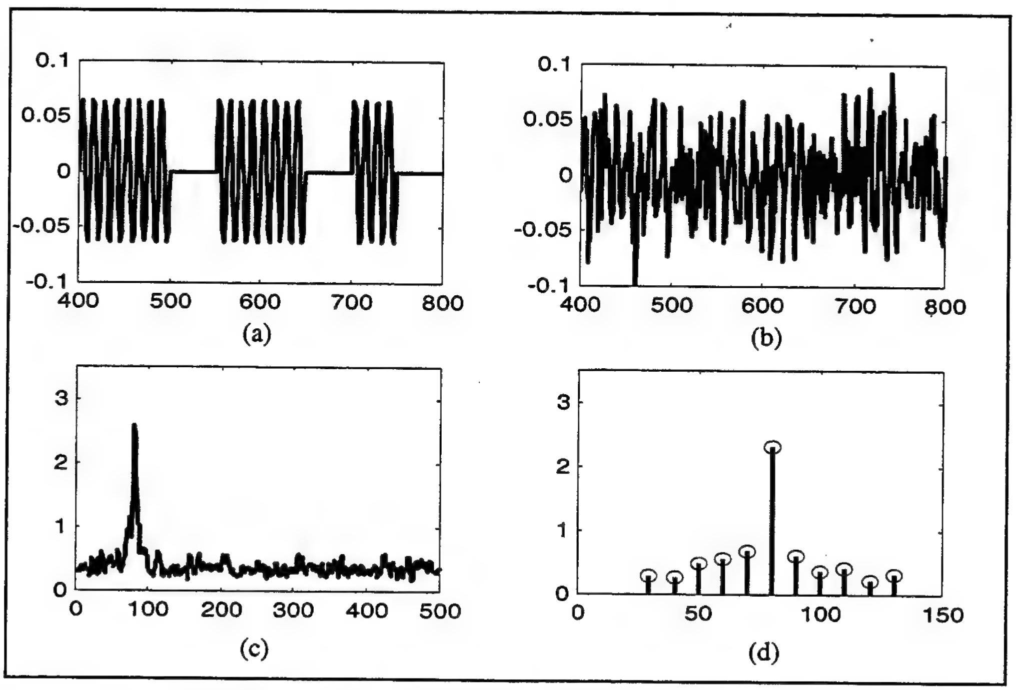


Figure V-7. 2-ASK Signal. (a) enlargement of modulated signal vs sample number (b) enlargement of corrupted signal vs sample number (SNR = 10 dB) (c) spectral characteristics vs frequency bin (d) extracted frequency bins.

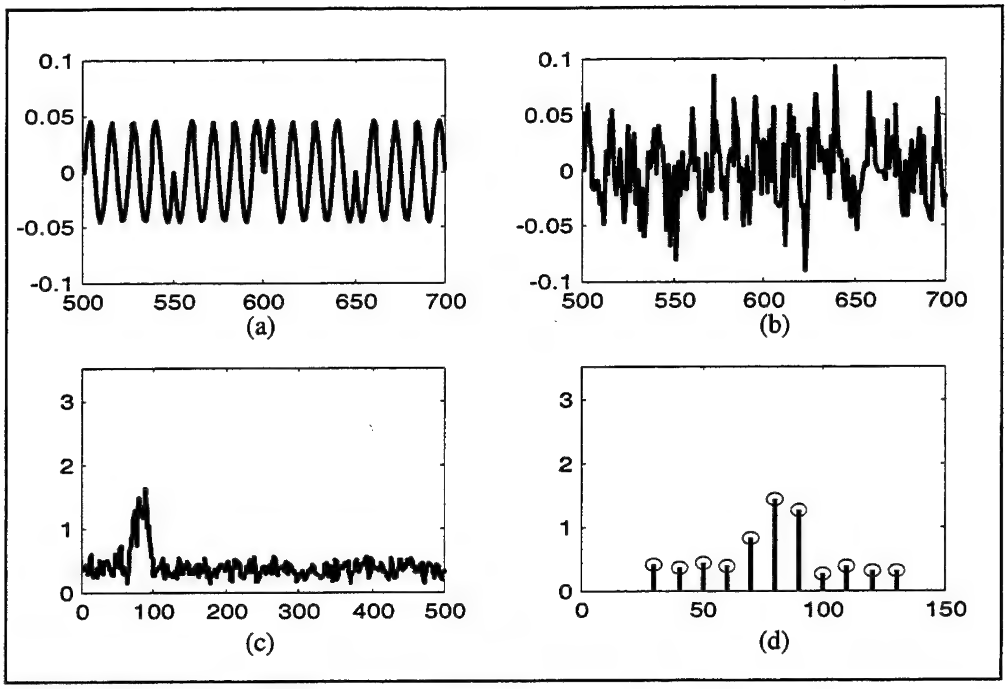


Figure V-8. 2-PSK Signal. (a) enlargement of modulated signal vs sample number (b) enlargement of corrupted signal vs sample number (SNR = 10 dB) (c) spectral characteristics vs frequency bin (d) extracted frequency bins.

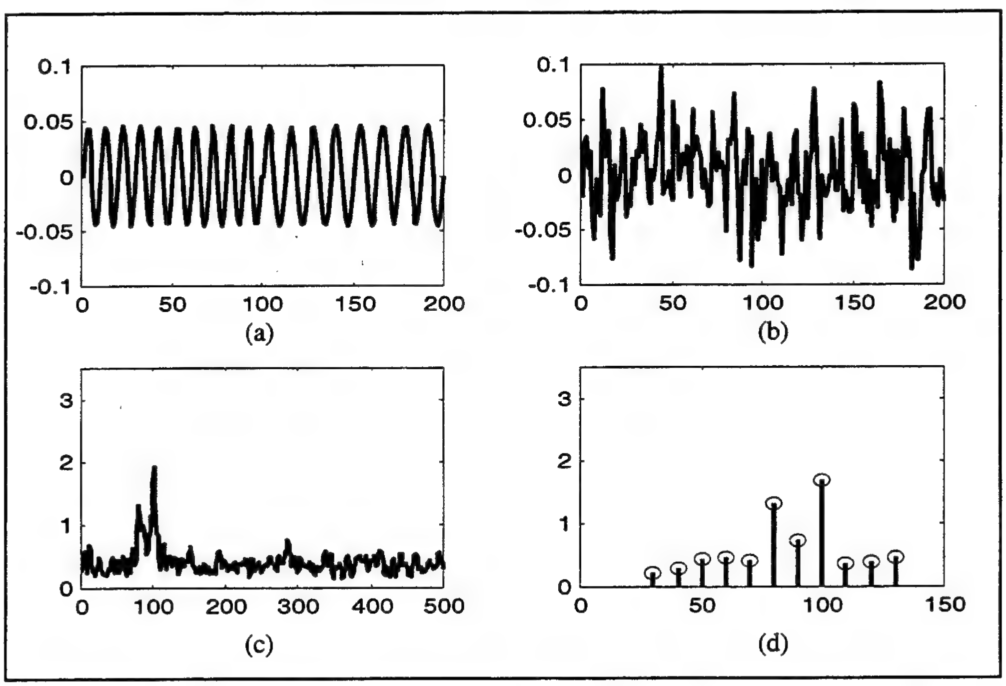


Figure V-9. 2-FSK Signal. (a) enlargement of modulated signal vs sample number (b) enlargement of corrupted signal vs sample number (SNR = 10 dB) (c) spectral characteristics vs frequency bin (d) extracted frequency bins.

environments would make for enhanced simulation realism. In addition, other digital signal types, such as radar, optical, and acoustic, could have been substituted for the ones implemented here. These factors can be explored in follow-on studies.

2. Simulation Protocol

The test procedure used to classify the simulated communication signals was the same as that used for the artificial signal features. Using the process described above, 100 training and 1000 testing data objects were created for each signal type per trial, with the set of simulation trials encompassing all combinations of SNR and signal space size. As before, these feature vectors were normalized (Equation 3.28) for use by the MSNN variant that required preconditioned input data (MSNN Mod 1) and the covariance matrices of the training observations were calculated for use by the statistical classifier.

The statistical classifier processed ten data sets of 300 training/3000 testing vectors each. For the neural networks, five sets of realizations were created; but because of neuro-classifier dependence on initial conditions, each data set was processed five times with varying starting weights and bias.

Section V.C reports the findings of these trials.

C. SIMULATION RESULTS

Results for the communication signal simulations are detailed in Appendix B, Tables B-19 through B-36 and Figures B-1 through B-6. For Tables B-19 through B-36, π_1, π_2, and π_3 refer to 2-ASK, 2-PSK, and 2-FSK, respectively.

Unlike the simulations conducted in Chapter IV, the no-noise case could be examined for the synthetic communication signals constructed. The results for these trials are included in Appendix B and summarized here in Table V-1. This table indicates that under no-noise conditions, the standard MSNN algorithm outperformed all other classifiers, with MSNN Mod 3 being almost as accurate. In particular, Table V-1 does not substantiate the improvements expected of the MSNN Mod 2 variant. It does, however, corroborate the Chapter IV findings of the MSNN Mod 1 variant. As before, the input preconditioning approach proved to be the least successful in classifying the generated signals. Chapter IV results also indicated that the statistical classifier most

successfully identified test objects. Table V-1, however, does not support this conclusion, showing instead that the quadratic classifier performed the least accurately.

CLASSIFICATION METHOD	11 FEATURES	26 FEATURES	51 FEATURES
Statistical Classifier	57.0	33.3	33.3
Perceptron	83.8	87.8	92.9
MSNN	94.3	93.8	94.8
MSNN Mod 1	45.1	64.4	63.0
MSNN Mod 2	91.4	92.2	92.8
MSNN Mod 3	92.6	93.1	94.0

Table V-1. Simulated Signal No-Noise Performance Results (Ave Percent Correct Classification).

To better understand the decline in statistical classifier performance as well as the results obtained with noise-corrupted signals, it is worthwhile to revisit Figures V-4 through V-9. Although the no-noise representation of these signals (Figures V-1 through V-3) clearly characterize the signal classes, the noise-corrupted plots show similarities in the feature descriptions of the different signal types, particularly between 2-ASK and 2-PSK. Comparing the 20 dB realizations of Figures V-4(d) and V-5(d), only the center frequency amplitudes differentiate the two modulation schemes. Coefficients of the remaining bins have approximately the same magnitude. When the 2-FSK signal is considered (Figure V-6(d)), the only significant distinction between the signal classes occur at bins 80 and 100, the two carrier frequencies of the 2-FSK modulation scheme. The same observations apply to the 10 dB examples.

Now considering Figures B-1 through B-6, the lack of distinguishing features between class types explains the poorer results obtained with the noise-corrupted simulated signal data as compared to the artificial features of Chapter IV. The reduced distinction between modulation types increased classifier confusion, thereby degrading

classification performance. Furthermore, altering the signal space dimension did not effect the average correct classification percentage of the MSNN variants suggesting that the information needed to separate the classes resided in a smaller number of features (Figures B-3 through B-6).

For the statistical classifier, the over-parameterized input space illustrates the *curse of dimensionality* (Bishop, 1995, p. 7). Unlike the neural classifiers that showed improved performance (albeit, marginal) with increased signal space size, the quadratic classifier exhibited poorer results (Figure B-1). These degraded results were attributed to ill-conditioning of the data matrix caused by a linear dependency of the chosen features. This supposition was verified by performing a principal component analysis (PCA) that reduced the feature space size (Bishop, 1995, p. 310-311). Doing so resulted in the improved classifier results of Table V-2.

FEATURES		**NO NOISE**		**SNR 20 dB**	
RETAINED	**INITIAL**	**BEFORE**	**AFTER**	**BEFORE**	**AFTER**
4	51	33.3	93.5	75.4	87.5
	26	33.3	93.1	79.6	87.0
	11	56.3	55.1	79.0	81.9
6	51	33.3	53.0	75.5	89.3
	26	33.3	44.5	79.3	87.1
	11	61.6	54.2	78.8	81.3

Table V-2. Statistical Classifier Performance Before and After Data Conditioning (Ave Percent Correct Classification).

Table V-2 confirms that the signal space was originally over-parameterized. In nearly all cases, the percentage of correct classifications increased, with significant gains observed in the no noise case for feature reductions from fifty-one and twenty-six to four. Only the no-noise, eleven-to-four or eleven-to-six reductions resulted in moderately

poorer results. The results obtained by the eleven-to-four component reduction can be attributed to statistical variance. It is expected that conducting more trials would effect no change due to data space conditioning. For the eleven-to-six reduction, the declining results are caused by selecting a basis set that increased the ambiguity between the distinct class data distributions, thereby incurring a loss of distinguishing information. But regardless of these instances, pre-processing of the input data through PCA techniques generally improved statistical classifier performance. Results validating this enhancement over all SNR conditions are included on Figure B-1.

Fortunately, the signal space over-parameterization that necessitated data pre-processing to obtain adequate statistical classifier performance has less effect on neural network accuracy. Granted, judicious feature extraction by methods such as principal component analysis improves neuro-classifier results; but intensive pre-processing is not essential since non-parametric classifiers let the "data speak for itself" (Haykin, 1994, p. 23). In addition, the over-parameterized feature space does not favor any particular neural network architectures and, hence, simulation results can be compared. Figures V-10 through V-12 compile the data of Figures B-1 through B-6 to provide this contrast of classifier capabilities.

Although all noise-corrupted simulated signal trials were inadequate based on the ninety-percent correct classification criteria stipulated in Chapter IV, Figures V-10 through V-12 does allow comparison of classifier performance. For instance, these graphs show that without input data conditioning by eigenvalue or other feature reduction techniques, the statistical classifier performed worse than all mean separator approaches except MSNN Mod 1 in the signal spaces considered. Using a principal component technique to reduce the input to four features, however, improved the statistical classifier accuracy to the same level as these MSNN methods.

Figures V-10 through V-12 also show that the perceptron performed worse than the MSNN variants in most cases. To account in part for this lower accuracy, Table V-3 lists the percentage of perceptron non-type classifications for each simulation trial. As

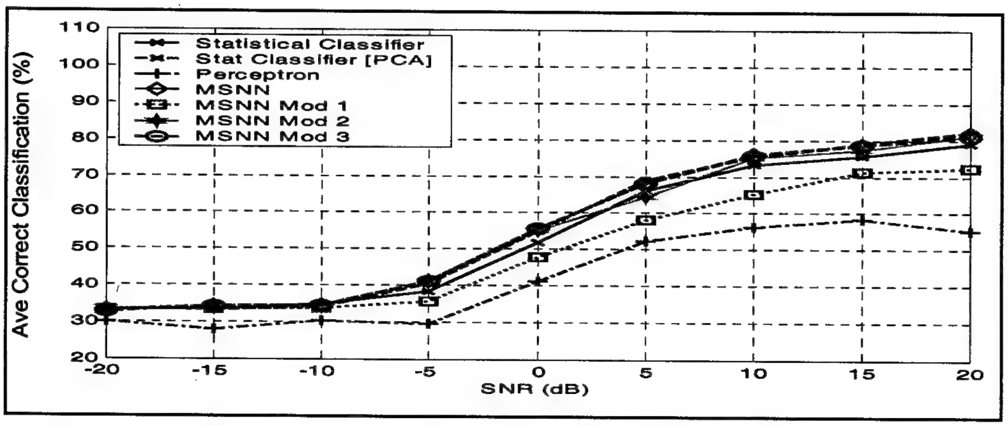


Figure V-10. Performance Comparison: Simulated Signals (11 features).

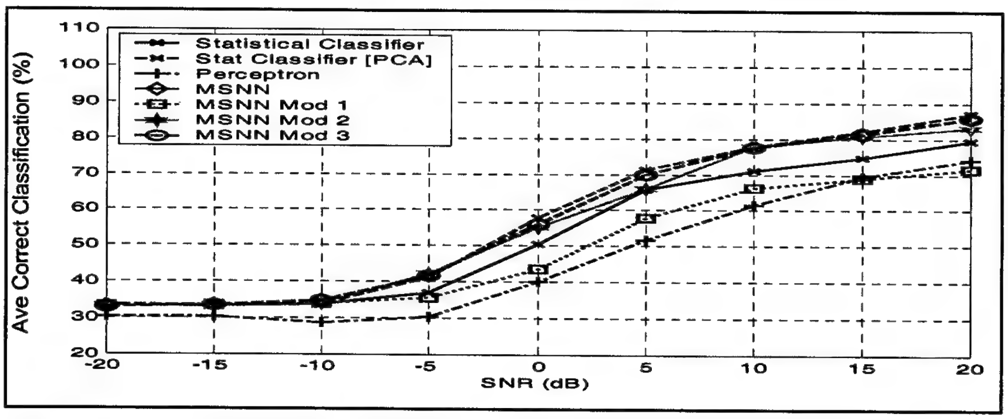


Figure V-11. Performance Comparison: Simulated Signals (26 features).

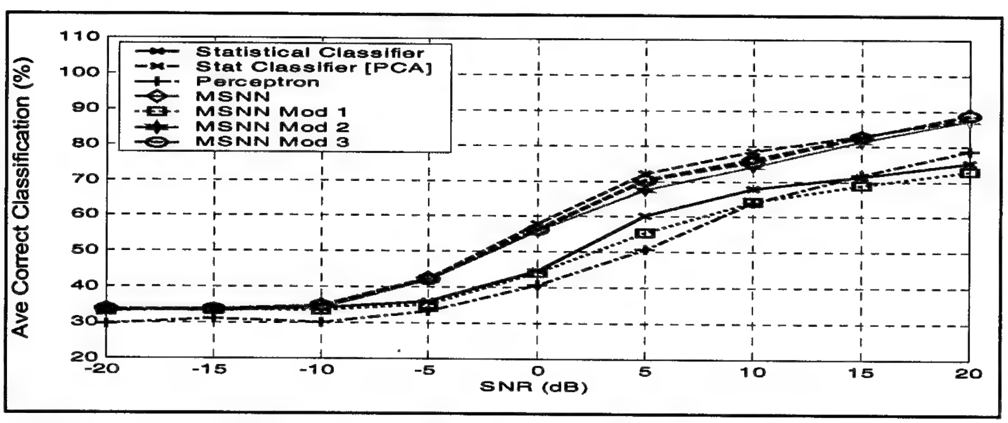


Figure V-12. Performance Comparison: Simulated Signals (51 features).

before, this poor classification performance by the perceptron is attributed to the neural network's inability to establish viable class separation.

SNR (dB)	11 FEATURES	26 FEATURES	51 FEATURES
No Noise	3.6	3.4	0.7
20	37	11.1	8.5
15	23.8	7.3	7.9
10	4.2	9.2	7.2
5	9.9	6.0	7.5
0	6.7	15.3	14.7
-5	15.0	16.8	10.7
-10	9.6	15.0	14.4
-15	17.5	8.8	6.9
-20	10.6	8.9	10.9

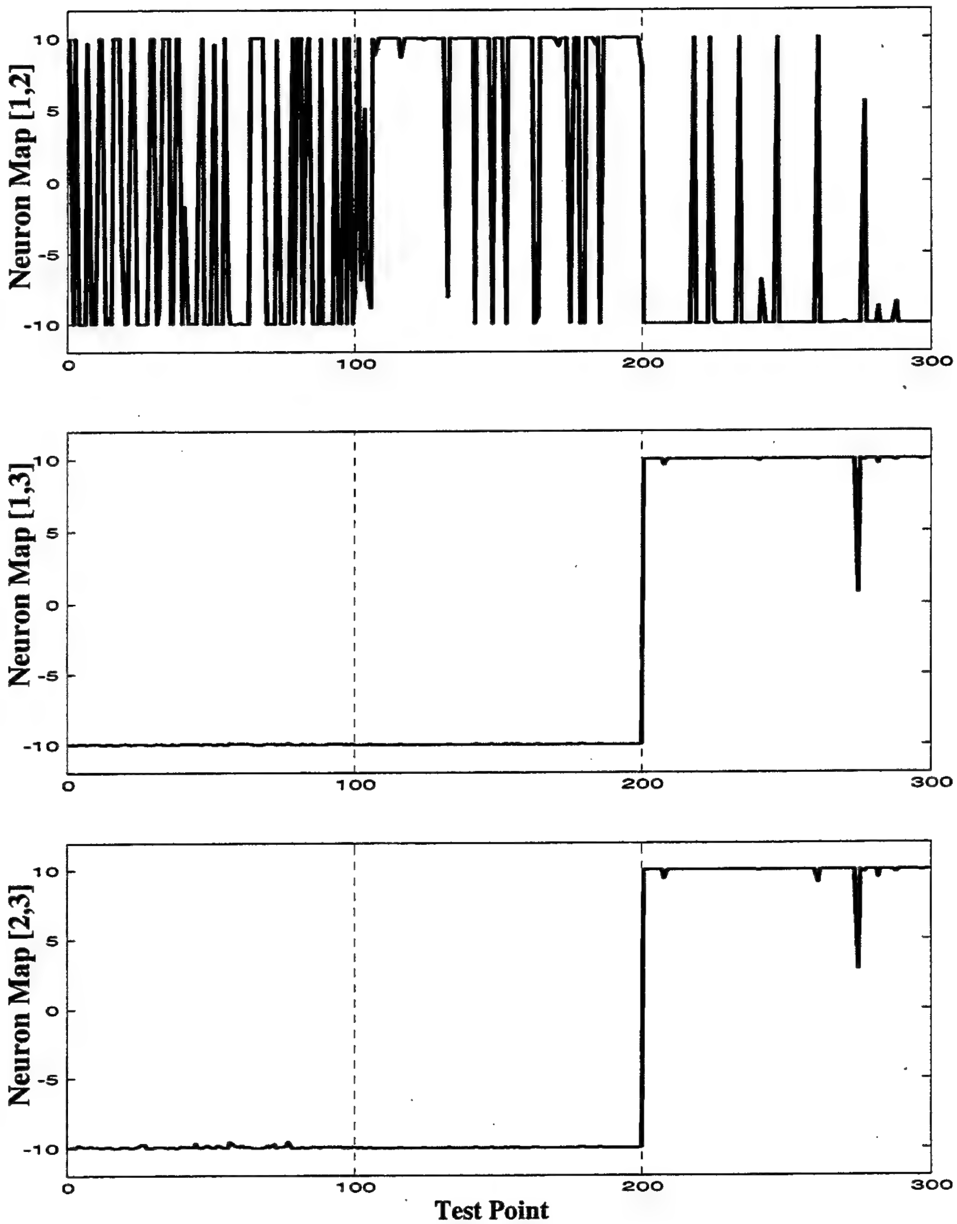
Table V-3. Observed Percentage of Perceptron Non-Type Classification.

In addition, these figures further substantiate the insufficiency of MSNN Mod 1. All plots show poorer performance for this MSNN variant as compared to the other MSNN techniques, with this degraded classification being attributed to the inherent similarity in the 2-ASK and 2-PSK signal descriptions and greater feature space data overlap resulting from input normalization.

With regards to the remaining MSNN variants, the outcome from trials conducted with noise-corrupted signals failed to conclusively identify which was more accurate. The simulation results were nearly identical. This, however, does not suggest a conceptual flaw in MSNN Mods 2 and 3, but rather indicates inadequate training. As before, network training for these modified techniques stopped on maximum epoch limit rather than satisfied VMR. Therefore, the networks were not effectively trained to

classify follow-on observation. Once more, increasing the epoch limit, refining the learning rate methodology, and softening of the VMR threshold may provided for MSNN performance distinction.

As final evidence of classifier performance, MSNN neuron maps for the SNR and feature space conditions of Figures V-4 through V-9 are provided. Shown as Figures V-13 through V-20, these plots support the findings just described. Of particular interest, Figures V-14 and V-18 demonstrate the inadequacy of MSNN Mod 1 by the non-uniformity of the neuron maps. In addition, the neuron maps for the remaining MSNN variants illustrate the similarity in 2-ASK and 2-PSK specifiers that resulted in equivalent performance plots.



2-ASK Specifier	2-PSK Specifier	2-FSK Specifier
-2.7	7.3	-8.8
-10	-10	9.9
-10	-10	9.9

Figure V-13. MSNN Neuron Map of 11-Features Simulated Signal Data (SNR = 20 dB).

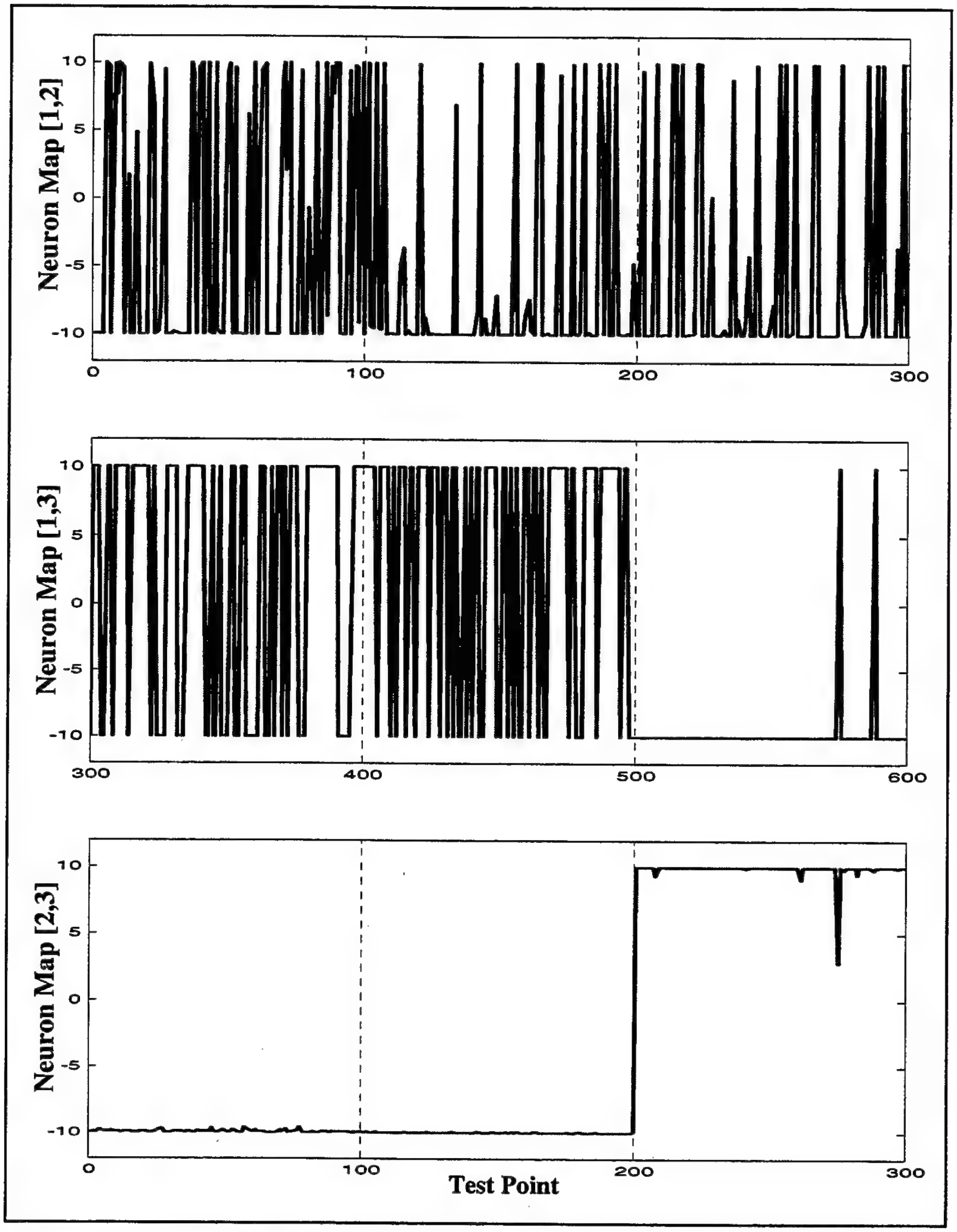
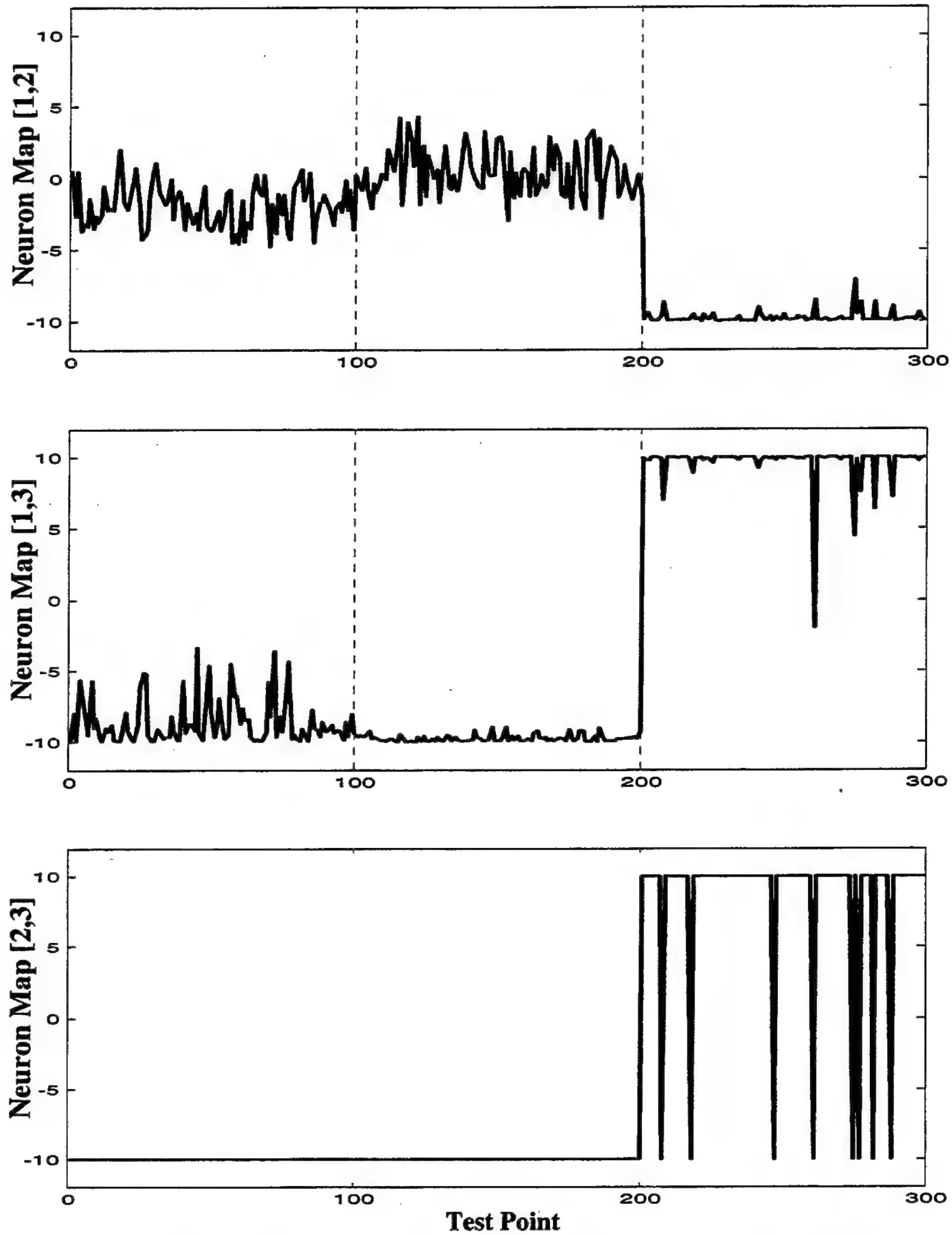
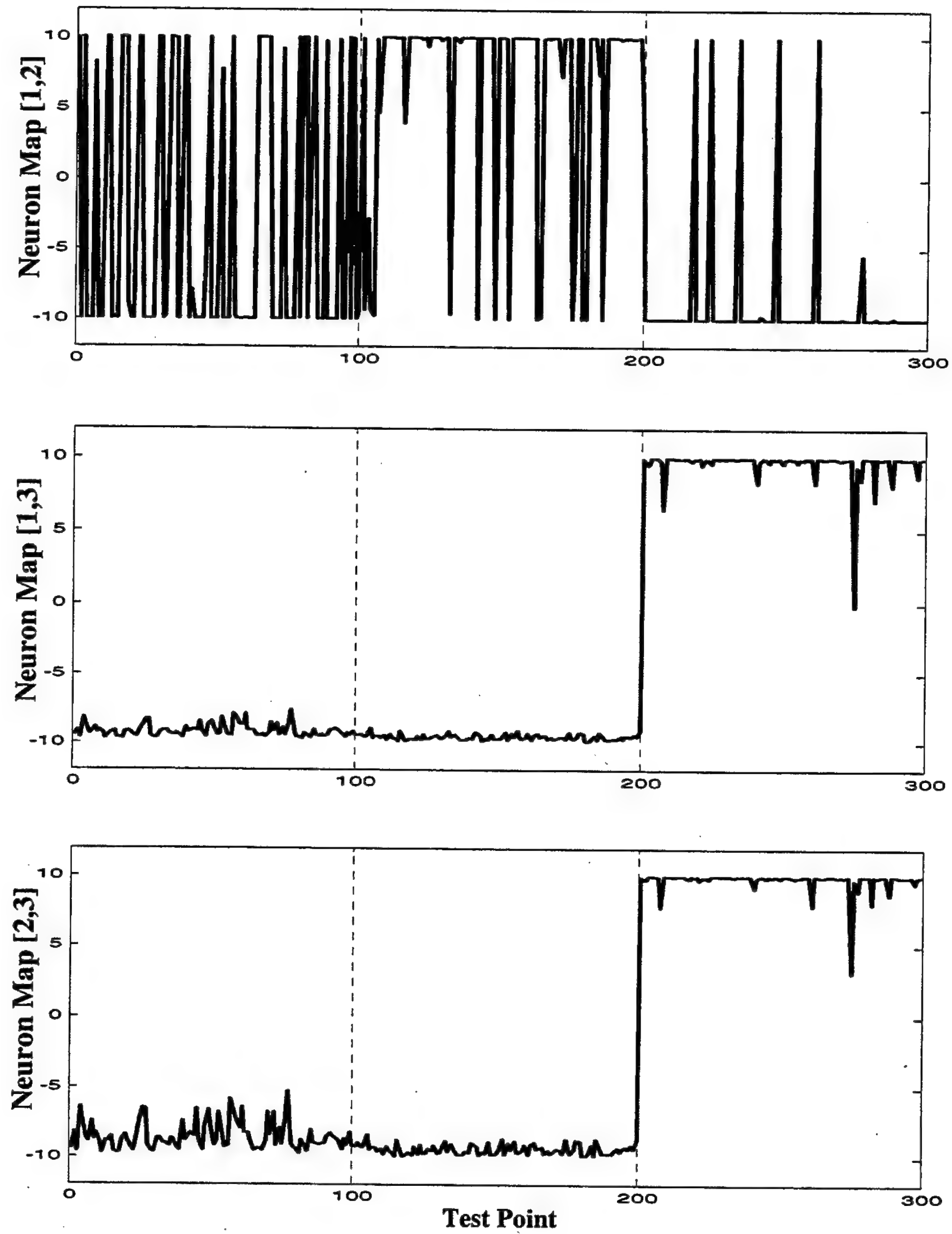


Figure V-14. MSNN Mod 1 Neuron Map of 11-Features Simulated Signal Data (SNR = 20 dB).



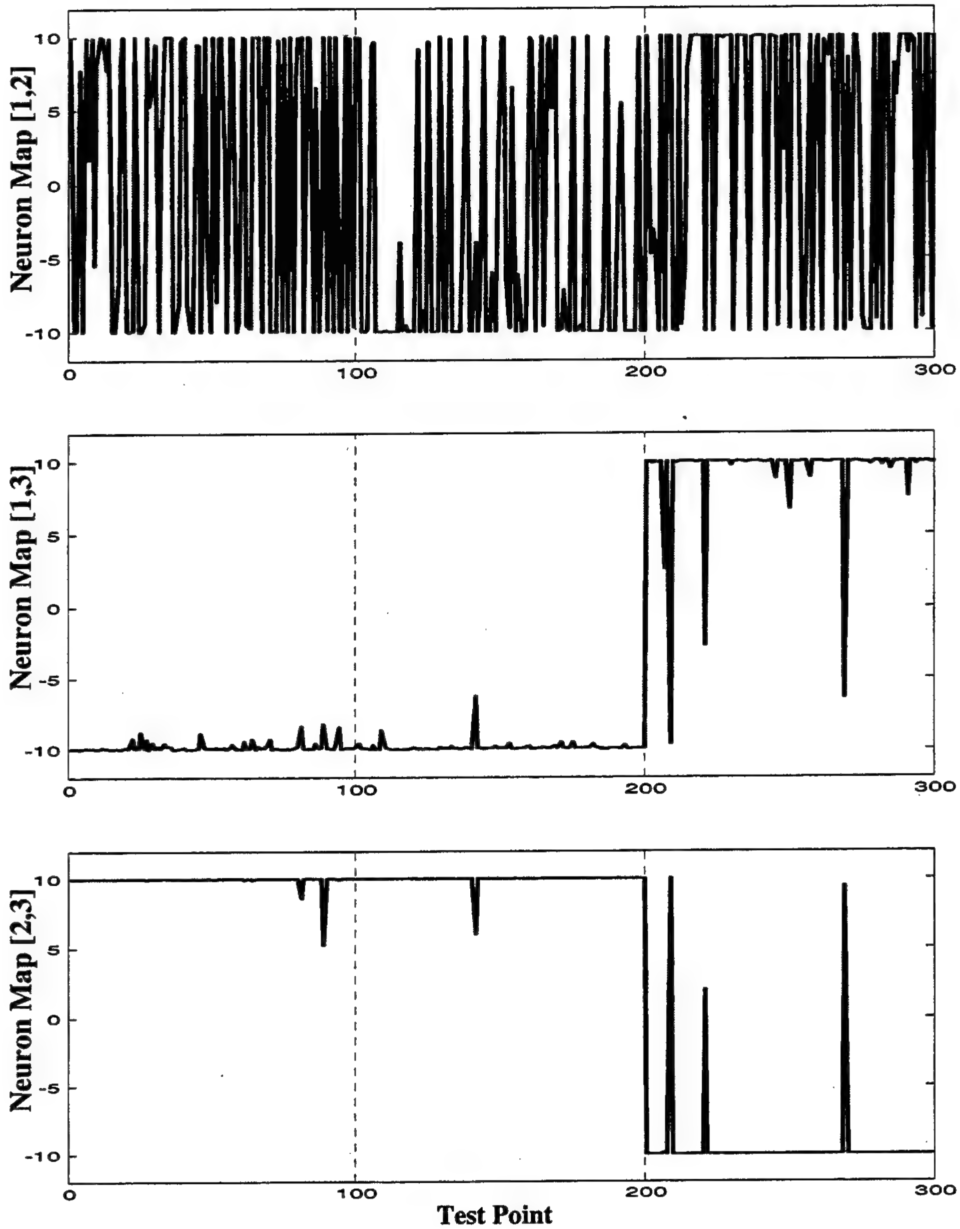
2-ASK Specifier	2-PSK Specifier	2-FSK Specifier
-1.9	0.4	-9.8
-8.6	-9.8	9.7
-10	-10	8.4

Figure V-15. MSNN Mod 2 Neuron Map of 11-Features Simulated Signal Data (SNR = 20 dB).



2-ASK Specifier	2-PSK Specifier	2-FSK Specifier
-3.0	6.9	-8.9
-9.2	-9.6	9.7
-8.5	-9.4	9.8

Figure V-16. MSNN Mod 3 Neuron Map of 11-Features Simulated Signal Data (SNR = 20 dB).



2-ASK Specifier	2-PSK Specifier	2-FSK Specifier
6.3	-4.6	3.3
-9.8	-9.9	9.3
9.9	10	-9.5

Figure V-17. MSNN Neuron Map of 11-Features Simulated Signal Data (SNR = 10 dB).

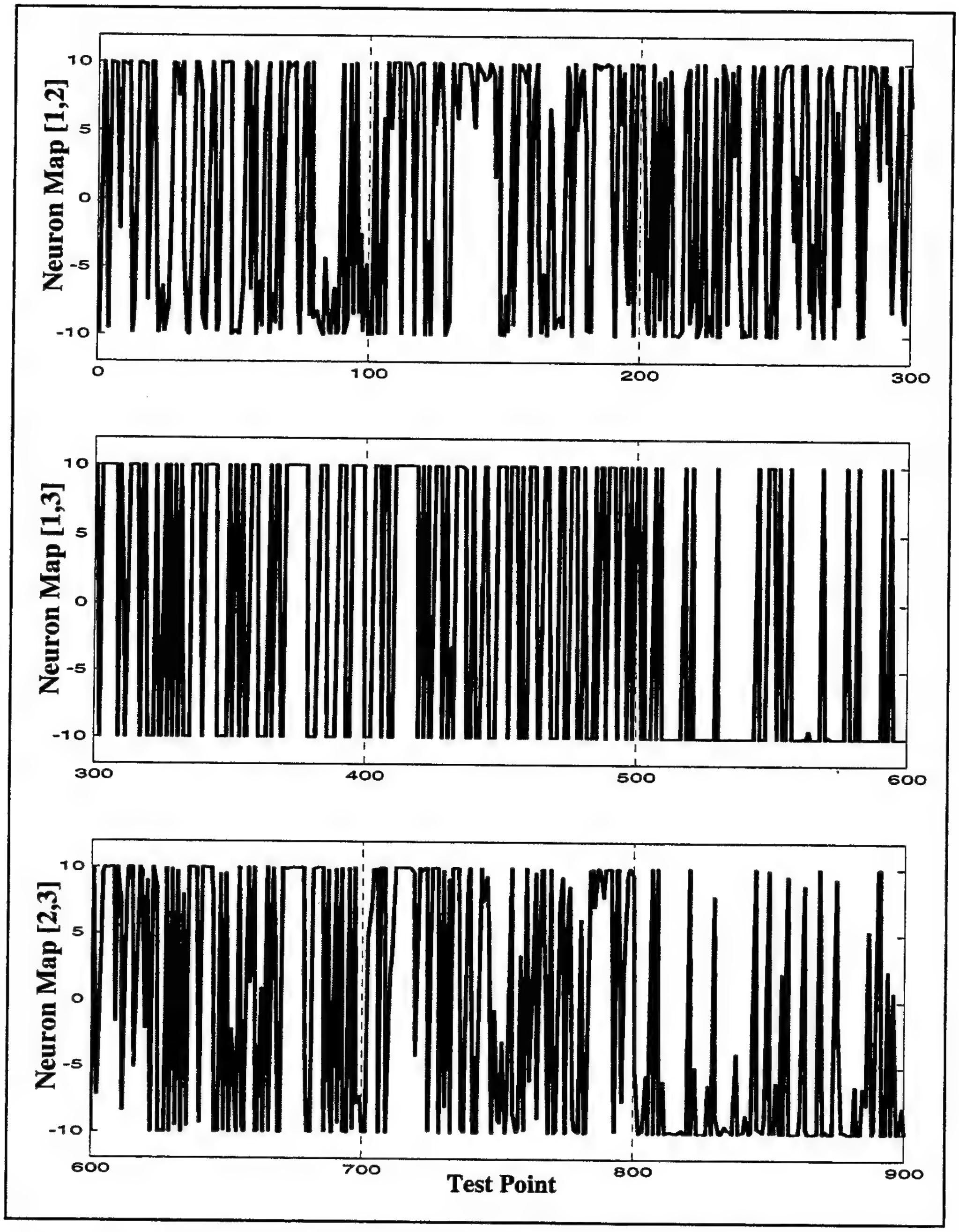
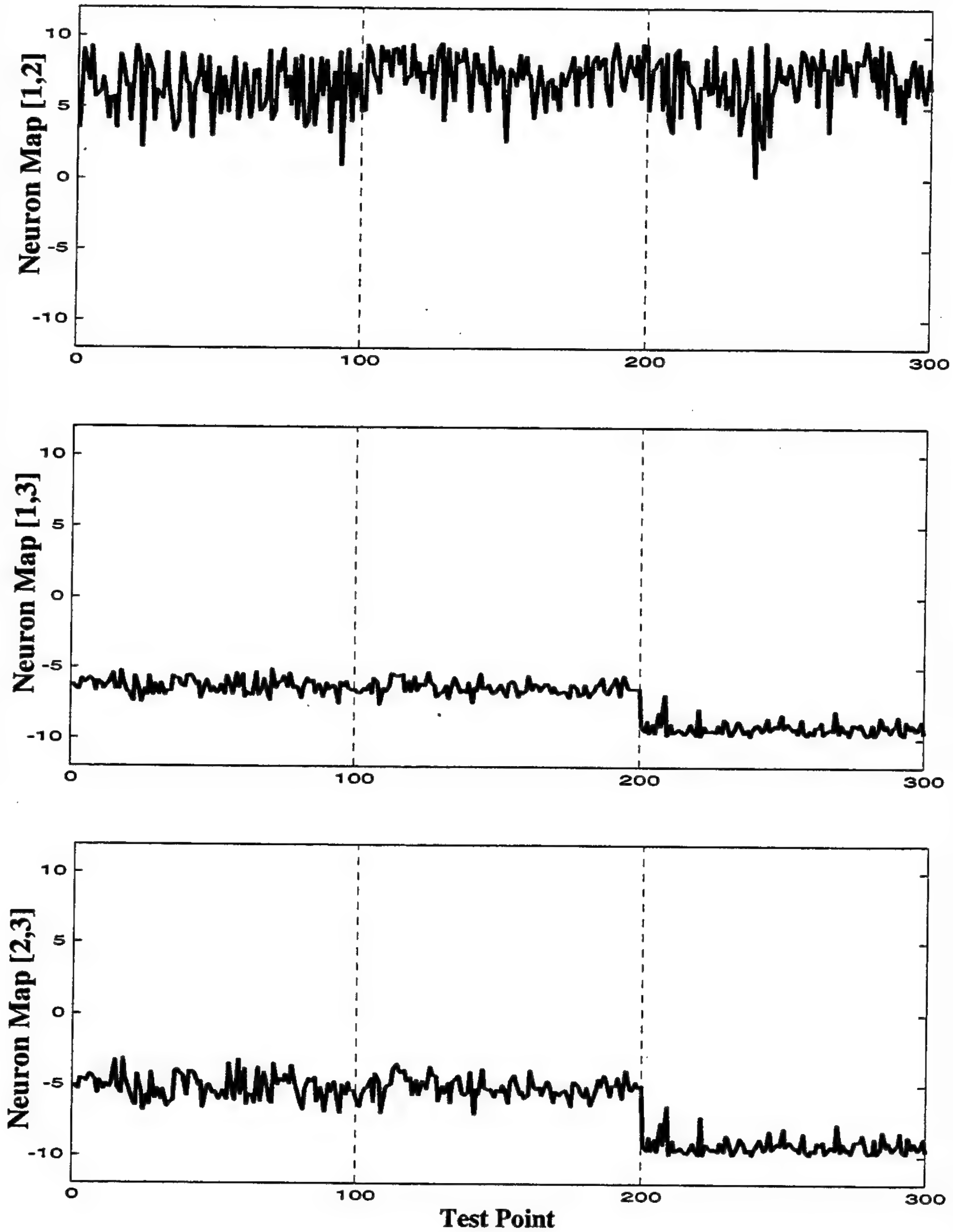
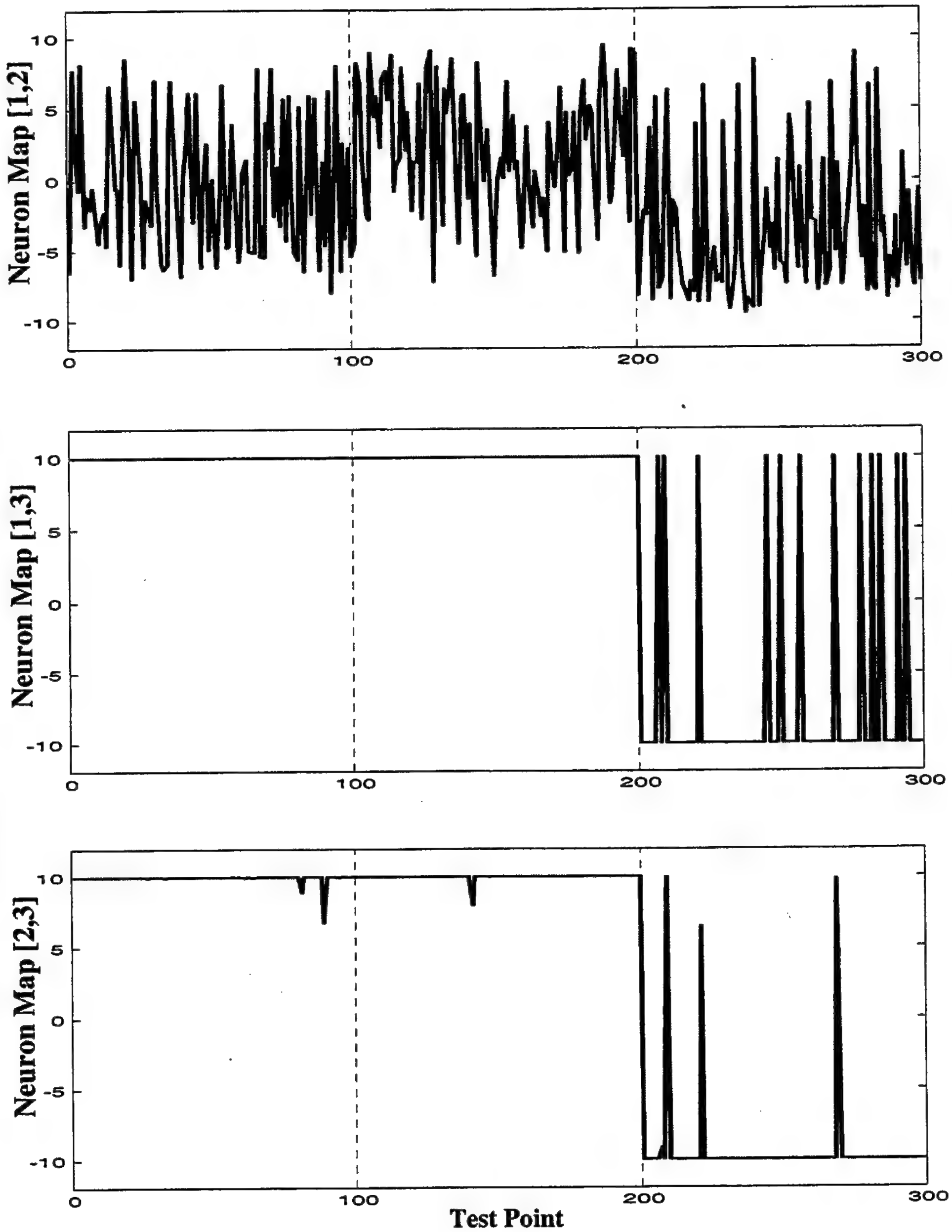


Figure V-18. MSNN Mod 1 Neuron Map of 11-Features Simulated Signal Data (SNR = 10 dB).



2-ASK Specifier	2-PSK Specifier	2-FSK Specifier
6.3	7.4	6.9
-6.4	-6.4	-9.2
-5.2	-5.2	-9.2

Figure V-19. MSNN Mod 2 Neuron Map of 11-Features Simulated Signal Data (SNR = 10 dB).



2-ASK Specifier	2-PSK Specifier	2-FSK Specifier
-5.3	2.2	-3.7
10	10	-7.6
10	10	-9.4

Figure V-20. MSNN Mod 3 Neuron Map of 11-Features Simulated Signal Data (SNR = 10 dB).

D. SUMMARY

Chapter V investigated the classification of software-generated communication signals in varying levels of noise. For the six classification methods discussed in this study, 100 testing and 1000 training realizations of 2-ASK, 2-PSK, and 2-FSK signals were created by encoding random binary messages. The experimental protocol followed the one used in Chapter IV. The quadratic classifier catalogued ten sets of data, while the neural networks processed only five. The neural networks, however, processed each data set five times from different initial conditions.

Figures V-10 through V-12 indicate that all trials were inaccurate (i.e., less than ninety-percent correct classification success). This observation, however, is not due to the classifiers themselves, but to the feature space definition. A more prudent selection would have included parameters that more distinctly differentiated the 2-ASK and 2-PSK signals. This not being the case, the simulation results showed a high degree of misclassification between these two modulation types.

Yet, the primary emphasis of this investigation was not to accurately categorize observations, but to compare classifier capabilities. For instance, analyzing noise-free signal data proved that the standard MSNN algorithm performed best. Furthermore, when considering noise-corrupted data, none of the proposed MSNN schemes showed substantial improvement over the standard approach. In particular, MSNN Mod 1 delivered inferior results due to the aforementioned feature description similarity in the 2-ASK and 2-PSK signals and increased data overlap caused by signal space normalization. The remaining MSNN methods produced outcomes comparable to the original MSNN formulation. Hence, no noteworthy advantage was realized by the proposed changes to the standard MSNN algorithm.

The MSNN techniques did fair markedly better than the perceptron neural network. Without *a priori* knowledge of the data set or optimal selection of signal features, the mean separators also performed better than the statistical classifier. Granted, when the input data was conditioned by feature space enhancing techniques such as the eigenvalue methods used here, dramatic gains in quadratic classifier performance were

realized. But, for the principal component reduction utilized, this improved outcome did not exceed the mean separator results, substantiating the greater utility of neural networks, in general, and the MSNN, in particular.

VI. CONCLUSIONS

A. SUMMARY OF WORK

The age of enhanced digital data collection and distribution requires electronic information management techniques that will assist and not hinder the warfighter. These applications must be rapid, reliable, and automated. This thesis investigated the continued development of one such tool.

The Mean Separator Neural Network (MSNN) had previously been applied to the classification of underwater signals. This study modified the MSNN and evaluated the performance of these variants in categorizing software simulated signals. Starting with a general introduction to neural networks, classification techniques were introduced and explained. In addition to the original MSNN developed by Duzenli and Fargues, two non-MSNN schemes were utilized as benchmarks to gauge proposed methods. The first considered was a pure parametric statistical classifier; specifically, a quadratic classifier. The decision rule for this statistical classifier was derived for later use.

The second benchmark implemented was a single layer perceptron neural network. The underlying concept of the perceptron was explained and its fundamental processing element constructed. In particular, the decision rule for perceptron neuro-classification was presented. To classify using the perceptron, however, first required training the network to discriminate the different class types. Hence, the perceptron learning rule and its role in network training was discussed. Finally, the disadvantages of the perceptron networks were identified as limitations due to the use of linear decision boundaries and the lack of solution optimization techniques. As an addendum, the Fixed-Increment Theorem of perceptrons was developed for edification. This precept specifies that for certain problem types, the perceptron neural networks will converge to a solution in a finite number of steps.

The central emphasis of this proof of concept study was enhanced implementation of the MSNN. But, to better understand these improvements, the standard MSNN classification scheme was first explained. The goal of the MSNN is to maximize the

mean separation of data projected into a decision space. The mathematical method for achieving this objective was presented as a basis for understanding the design of the MSNN neural processing element. Then, using this fundamental building block, the study next examined the three stages of solving a classification problem with the MSNN: training, typing, and decision-making.

Network training was accomplished using a steepest descent algorithm in which the training trajectory was governed by the mean-difference projection index, MD. This training algorithm also employed a dynamic learning rate rule to control the training trajectory.

After training the network, typing was completed by using the mean separator equation to assign a unique numerical sequence to each class. In the decision-making stage, these class specifiers are then compared to the network output of subsequent data to identify the uncategorized observations.

By merely focusing on maximizing mean separation, however, the MSNN fails to recognize the impact of data variance. Indeed, wide mean separation may be inconsequential if data spread is equally large. Conversely, a small difference in projection space means could be acceptable for tightly clustered data. Because of this, three modifications to the MSNN algorithm were proposed and evaluated.

The first MSNN variant (MSNN Mod 1) suggested that MSNN performance may be improved by pre-processing the input data. By normalizing the data about its mean, we endeavored to tighten the input data distribution and reduce data overlap in the feature space. Mapping these distributions into the decision space would then result in greater precision to the optimal values; thus, less intersection of the decision space distributions and greater classification accuracy. Unfortunately, it was recognized that this may not be the case. Input data normalization may increase input data diffusion and transformation into the decision space may not preserve cluster cohesion.

The second MSNN variant (MSNN Mod 2) sought to improve mean separator performance by normalizing the projection space instead of the input space. Essentially, the concept entailed maximizing projection data mean spread relative to projection data

variance. Doing this provides for thorough evaluation of the projection data distributions. Because of this, a large mean separation may or may not be beneficial dependent on how accurately the input data was mapped into the decision space. That is, data projection resulting in a large mean difference may be meaningless if the projected data variance was also significantly large. Conversely, small mean separation could be tolerable for instances of small data variance.

Implementation of this model, however, was not as straightforward as that of the pre-conditioned input variant. Whereas the pre-conditioned input data method only required normalizing the feature space and adjusting the decision scheme, accounting for projection space variance necessitated deriving a new performance index (MD_2) and training termination parameter (VMR).

The third MSNN variant (MSNN Mod 3) investigated utilized the projection index of the standard MSNN algorithm coupled with the new training termination parameter developed for the normalized projection space method.

Utilizing these six classification methods, two types of trials were conducted. In the first, random vectors composed of simulated feature components were generated. Classifier performance, reported as a percentage value, was measured as the accuracy obtained in properly categorizing test data of known class type. In general, increased SNR and feature space dimensionality produced improved classifier performance for all techniques. Of the benchmarks used, the statistical classifier had the best classification results; the perceptron, the worse for all but the largest feature space trials.

The MSNN variants produced inconclusive results. MSNN Mod 1 performed markedly worse with a small feature space size. But as feature space dimensionality became larger, input data preconditioning delivered significantly better results. The classification performance of MSNN Mod 2 equaled that of the standard MSNN approach. This lack of significant improvement was predominately due to the MSNN Mod 2 networks not being adequately trained. Network training often terminated on maximum epoch cycles rather than VMR threshold. This same reason also partly explains the classification performance of MSNN Mod 3.

Having gained a rudimentary understanding of each classifier's capabilities, a second set of trials tested their performance with software simulated communication signals. Specifically, three types of binary modulation schemes were implemented: 2-ASK, 2-PSK, and 2-FSK.

As before, the perceptron had the worse classification results. The statistical classifier, however, did not demonstrate the best performance. In fact, unlike the other techniques, the quadratic classifier showed lower accuracy with increased feature space size. This tendency was due to a correlation between feature space components, resulting in an ill-conditioned covariance matrix. Extracting the principal components to reduce the input dimensionality dramatically improved statistical classifier performance.

Examining the outcome of no-noise trials, the standard MSNN methodology outperformed all other classifiers. Moreover, when considering noise-corrupted signals, simulation results were, as in Chapter IV, irresolute. MSNN Mod 1 did consistently present the worse results, presumably due to the similarity in 2-ASK and 2-PSK feature components and increased signal space data diffusion caused by normalization. All other methods were essentially equivalent. The lack of improvement from MSNN Mods 2 and 3 was ascribed to inadequate network training.

B. SUGGESTIONS FOR FUTURE RESEARCH

The intent of this thesis was to propose and validate modifications to the MSNN classifier. Three such modifications were presented. When considering noise-corrupted signals, none showed significant improvement over the standard MSNN approach. In particular, MSNN Mod 2, which emphasized projection data variance in addition to mean separation, only performed as well as the standard MSNN algorithm. This lack of proof of concept, however, is not due to discrepancies in the underlying fundamentals of the approach, but rather to method implementation. In particular, two aspects deserve further consideration.

One likely cause of inadequate network training using the MSNN Mod 2 variant may be due to reaching the maximum epoch limit prior to satisfying the VMR threshold. Therefore, to improve the performance of the MSNN projection space normalization

scheme, the maximum epoch setpoint and learning rate rules require thorough investigation. With regards to the latter, instead of using an adaptive learning rate approach, starting with a static learning rate (i.e., one that is only dependent on the gradient of the performance parameter) may provide better results when compared to the standard mean separator.

In addition, it may also be instructive to reduce the stringency of the VMR threshold. As used in this study, a VMR value of zero equates to 0.5% class overlap, assuming normally distributed data. Furthermore, the termination requirement sets the VMR threshold at 0.90. This combination of overlap and ratio may be unnecessarily restrictive. Therefore, studies could be conducted to empirically establish justifiably values.

The termination requirement for MSNN Mod 2 should also be re-evaluated. VMR was used as a training terminator only if the projection index (MD_2 for MSNN Mod 2 and MD for MSNN Mod 3) showed training movement towards an improved solution. For MSNN Mod 2 this would have become apparent in the VMR value itself. Therefore, the requirement to show decreasing performance parameter values is unnecessary. For MSNN Mod 3, the performance parameter only takes into account projected data mean separation. By neglecting to consider data variance, the underlying principle of VMR is disregarded since improved conditions could result when mean separation decreases (provided the relative decrease in data variance is greater).

Because of this inadequacy of MSNN Mod 3, it may have been more beneficial to use VMR as the performance parameter instead of either of the two mean-difference equations. This performance parameter would essentially be the reciprocal of MD_2. As such, the difficulties encountered due to the infinitesimally small projection variances (i.e., division by zero) would be avoided.

Once an optimal mean separator algorithm has been determined, the modified MSNN classifier could be used to identify real-world signals (e.g., radar, communication, acoustic). This would, however, require a high degree of classifier accuracy. Recall that the intent of this investigation was to compare proposed alterations to the MSNN

algorithm. As such, absolute classifier accuracy was not the aim; rather relative classifier accuracy was. If a high degree of absolute classifier accuracy is desired (such as for categorizing real-world signals), judicious feature extraction schemes and pre-processing techniques are needed. When proved successful, the modified MSNN classifier utilizing this refined feature selection approach can then be expanded from a software application to direct implementation on an integrated circuit. Having such a device would greatly aid the operational commander in understanding the battlespace and making critical decisions.

APPENDIX A.FIXED-INCREMENT CONVERGENCE THEOREM

Rosenblatt reasoned that for a single-layer perceptron applied to linear separable problems, a solution can be determined in a finite number of iterations. Stated formally, this *fixed-increment convergence theorem* asserts:

> Let the subsets of training vectors X_1 and X_2 be linearly separable. Let the inputs presented to the single-layer perceptron originate from these two subsets. The perceptron converges after some n_0 iterations, in the sense that
>
> $$\mathbf{w}(n_0) = \mathbf{w}(n_0 + 1) = \mathbf{w}(n_0 + 2) = \ldots$$
>
> is a solution vector for $n_0 \leq n_{max}$. (Haykin, 1994, p.111).

To prove this theorem, the following vector notation is used for convenience:

$$\mathbf{x} = \begin{bmatrix} \mathbf{w} \\ b \end{bmatrix} \text{ and } \mathbf{z} = \begin{bmatrix} \mathbf{p} \\ 1 \end{bmatrix}. \tag{A.1}$$

Using this notation, the input to the *hardlim* activation function, n, can be expressed as

$$n = \mathbf{w} \bullet \mathbf{p} + b = \mathbf{x}^T \bullet \mathbf{z}. \tag{A.2}$$

Similarly, the perceptron learning rule Equations 3.11 and 3.12 can be combined into the single vector equation

$$\mathbf{x}^{new} = \mathbf{x}^{old} + e\mathbf{z}. \tag{A.3}$$

Given a solution $\mathbf{x}^*$ to the classification problem,

$$n = \mathbf{x}^{*T} \bullet \mathbf{z} \begin{cases} > \delta > 0 & \text{if } t = 1 \\ < -\delta < 0 & \text{if } t = 0 \end{cases} \tag{A.4}$$

Equation A.4 implies that there exists a positive δ less than the magnitude of the inner product n for both target output possibilities.

After k training iterations, the perceptron learning rule (Equation A.3) results in an updated solution be given by

$$\mathbf{x}(k) = \mathbf{z}'(k-1) + \mathbf{z}'(k-2) + \ldots + \mathbf{z}'(0), \tag{A.5}$$

where the prime (′) accounts for the possible error values 0 and ±1 and it is assumed that the $\mathbf{w}(0) = 0$. Taking the inner product of the solution vector $\mathbf{x}^*$ with Equation A.5 yields

$$\mathbf{x}^{*T} \bullet \mathbf{x}(k) = \mathbf{x}^{*T} \bullet \mathbf{z}'(k-1) + \mathbf{x}^{*T} \bullet \mathbf{z}'(k-2) + \ldots + \mathbf{x}^{*T} \bullet \mathbf{z}'(0) \tag{A.6}$$

and using the inequality relationships of Equation A.4 in Equation A.6 leads to

$$\mathbf{x}^{*T} \bullet \mathbf{x} > k\delta, \tag{A.7}$$

with δ chosen as the minimum $\mathbf{z}'(i)$. With the Cauchy-Schwartz inequality, a lower bound on the square of the weight vector $\mathbf{x}(k)$ is therefore found to be

$$\|\mathbf{x}(k)\|^2 \geq \frac{(\mathbf{x}^{*T} \bullet \mathbf{x}(k))^2}{\|\mathbf{x}^*\|^2} > \frac{(k\delta)^2}{\|\mathbf{x}^*\|^2} \tag{A.8}$$

To find the upper bound for the square of the weight vector at iteration k, Equation A.3 is substituted into the length equation:

$$\begin{aligned} \|\mathbf{x}(k)\|^2 = \mathbf{x}^{*T}(k) \bullet \mathbf{x}(k) &= [\mathbf{x}(k-1) + \mathbf{z}'(k-1)]^T \bullet [\mathbf{x}(k-1) + \mathbf{z}'(k-1)] \\ &= \|\mathbf{x}(k-1)\|^2 + \|\mathbf{z}'(k-1)\|^2 + 2\mathbf{x}^T(k-1)\mathbf{z}'(k-1) \end{aligned} \tag{A.9}$$

When proper classification occurs, the cross-term in Equation A.9 will be zero. If misclassification occurs, this term will be negative. Hence, Equation A.9 can be rewritten as an inequality:

$$\|\mathbf{x}(k)\|^2 \leq \|\mathbf{x}(k-1)\|^2 + \|\mathbf{z}'(k-1)\|^2. \tag{A.10}$$

Repeating this derivation for all previous iterations of $\|\mathbf{x}(i)\|^2$, the upper bound on the square of the weight vector is found to be

$$\|\mathbf{x}(k)\|^2 \leq \|\mathbf{z}'(0)\|^2 + \|\mathbf{z}'(1)\|^2 + \cdots + \|\mathbf{z}'(k-1)\|^2 \leq k\Delta \tag{A.11}$$

where Δ is the maximum $\mathbf{z}'(i)$.

Finally, combining Equations A.8 and A.11 results in a closed form solution for the number of iterations, k, required for perceptron convergence:

$$\frac{(k\delta)^2}{\|\mathbf{x}^*\|^2} \le \|\mathbf{x}(k)\|^2 \le k\Delta$$

$$k \le \frac{\Delta\|\mathbf{x}^*\|^2}{\delta^2} \tag{A.12}$$

The assumptions made to arrive at this conclusion were that (1) a solution is known to exist and (2) the length of the input vectors is upper-bounded. (Hagan, et al, 1996, pp. 4-15 – 4-18).

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B. SIMULATION RESULTS

To determine the capabilities of the classifiers studied, two types of simulations were conducted. The first set of tests gauged the performance of the different classifiers by creating artificial features for different class types. Once provided with this initial assessment of the different classification schemes, the second simulation measured their ability to categorize simulated communication signals. Appendix B contains the results from both types of trials.

Simulation results are presented in two forms. On Tables B-1 through B-36, confusion matrices report classifier performance. A *confusion matrix* is an m x m matrix, m being the number of categories in the classification problem. Read horizontally, each confusion matrix lists the correct class type; vertically, the class type selected by the classifier. The elements within each matrix indicate the percentage of objects (i.e., testing input data vectors) categorized as a certain class. In particular, the diagonal elements give the percentage of correct classifications for a particular simulation situation. Averaging these diagonal elements results in a performance index for that particular classifier under the specified conditions. Disregarding slight deviation due to round-off error, each table row sums to 100% for all classifiers except the perceptron neural network. The confusion matrices for the perceptron neural networks do not show rows that sum to 100% due to non-class typings as reported on Tables IV-1 and V-2.

Tables B-1 through B-18 report results for the first set of simulations conducted; classification of data objects consisting of artificial features. Tables B-19 through B-36 report results for the set of simulations conducted on simulated communication signals. On these latter tables, π_1, π_2, and π_3 correspond to simulated 2-ASK, 2-PSK, and 2-FSK class of software created signals.

Plots of the average performance indices permit visual analysis of the effect of varying noise level and input space dimensionality. These graphs are provided as Figures B-1 through B-6. Chapters IV and V contain performance index graphs that allow direct comparison of the different classification methods.

For all tables and plots, MSNN Mod 1 refers to the MSNN variant with input space preconditioning; MSNN Mod 2, the MSNN variant with projection space normalization; and MSNN Mod 3, the MSNN variant utilizing the standard MSNN performance parameter, MD, and the new training termination limit, VMR.

INDEX: 99.1		SELECTED		
		π_1*	π_2*	π_3*
ACTUAL	π_1	**99.8**	0.2	0.0
	π_2	0.2	**98.2**	1.7
	π_3	0.0	0.6	**99.4**

SNR = 20 dB

INDEX: 97.7		SELECTED		
		π_1*	π_2*	π_3*
ACTUAL	π_1	**98.5**	0.3	1.2
	π_2	0.4	**97.8**	1.8
	π_3	1.4	1.7	**96.8**

SNR = 15 dB

INDEX: 94.0		SELECTED		
		π_1*	π_2*	π_3*
ACTUAL	π_1	**92.6**	5.8	1.6
	π_2	6.3	**92.3**	1.4
	π_3	1.6	1.3	**97.1**

SNR = 10 dB

INDEX: 86.2		SELECTED		
		π_1*	π_2*	π_3*
ACTUAL	π_1	**92.0**	3.5	4.5
	π_2	4.2	**83.5**	12.3
	π_3	5.6	11.4	**83.1**

SNR = 5 dB

INDEX: 73.2		SELECTED		
		π_1*	π_2*	π_3*
ACTUAL	π_1	**74.2**	11.1	14.7
	π_2	8.5	**83.4**	8.1
	π_3	18.2	19.7	**62.0**

SNR = 0 dB

INDEX: 64.6		SELECTED		
		π_1*	π_2*	π_3*
ACTUAL	π_1	**63.9**	17.7	18.4
	π_2	18.2	**61.5**	20.3
	π_3	16.6	15.1	**68.4**

SNR = -5 dB

INDEX: 58.8		SELECTED		
		π_1*	π_2*	π_3*
ACTUAL	π_1	**64.3**	16.8	18.9
	π_2	20.9	**66.2**	12.9
	π_3	31.1	23.0	**45.8**

SNR = -10 dB

INDEX: 52.7		SELECTED		
		π_1*	π_2*	π_3*
ACTUAL	π_1	**53.7**	21.3	25.1
	π_2	26.2	**46.2**	27.6
	π_3	25.5	16.1	**58.4**

SNR = -15 dB

INDEX: 60.1		SELECTED		
		π_1*	π_2*	π_3*
ACTUAL	π_1	**59.6**	23.8	16.6
	π_2	20.4	**57.4**	22.2
	π_3	15.8	20.8	**63.4**

SNR = -20 dB

Table B-1. Confusion Matrices for Simulated Feature Trials (Three-Class, Three-Features): Statistical Classifier. (see App B cover page for table description)

INDEX: 100	SELECTED		
	π_1*	π_2*	π_3*
ACTUAL π_1	**100**	0.0	0.0
ACTUAL π_2	0.0	**100**	0.0
ACTUAL π_3	0.0	0.0	**100**

SNR = 20 dB

INDEX: 100	SELECTED		
	π_1*	π_2*	π_3*
ACTUAL π_1	**100**	0.0	0.0
ACTUAL π_2	0.0	**100**	0.0
ACTUAL π_3	0.0	0.0	**100**

SNR = 15 dB

INDEX: 100	SELECTED		
	π_1*	π_2*	π_3*
ACTUAL π_1	**100**	0.0	0.0
ACTUAL π_2	0.0	**100**	0.0
ACTUAL π_3	0.0	0.0	**100**

SNR = 10 dB

INDEX: 99.7	SELECTED		
	π_1*	π_2*	π_3*
ACTUAL π_1	**99.7**	0.3	0.1
ACTUAL π_2	0.4	**99.5**	0.1
ACTUAL π_3	0.0	0.0	**99.9**

SNR = 5 dB

INDEX: 96.7	SELECTED		
	π_1*	π_2*	π_3*
ACTUAL π_1	**96.6**	2.6	0.8
ACTUAL π_2	2.3	**95.6**	2.1
ACTUAL π_3	0.4	1.8	**97.7**

SNR = 0 dB

INDEX: 89.1	SELECTED		
	π_1*	π_2*	π_3*
ACTUAL π_1	**90.8**	6.1	3.1
ACTUAL π_2	6.8	**87.3**	5.9
ACTUAL π_3	4.0	6.7	**89.3**

SNR = -5 dB

INDEX: 76.5	SELECTED		
	π_1*	π_2*	π_3*
ACTUAL π_1	**79.6**	13.7	6.7
ACTUAL π_2	13.2	**72.5**	14.3
ACTUAL π_3	8.0	14.4	**77.5**

SNR = -10 dB

INDEX: 73.3	SELECTED		
	π_1*	π_2*	π_3*
ACTUAL π_1	**83.8**	8.4	7.7
ACTUAL π_2	12.6	**66.1**	21.3
ACTUAL π_3	11.7	18.4	**69.9**

SNR = -15 dB

INDEX: 77.1	SELECTED		
	π_1*	π_2*	π_3*
ACTUAL π_1	**75.6**	15.8	8.6
ACTUAL π_2	13.7	**76.5**	9.8
ACTUAL π_3	10.3	10.5	**79.1**

SNR = -20 dB

Table B-2. Confusion Matrices for Simulated Feature Trials (Three-Class, Ten-Features): Statistical Classifier. (see App B cover page for table description)

INDEX: 100		SELECTED π_1^*	π_2^*	π_3^*
ACTUAL	π_1	**100**	0.0	0.0
	π_2	0.0	**100**	0.0
	π_3	0.0	0.0	**100**

SNR = 20 dB

INDEX: 100		SELECTED π_1^*	π_2^*	π_3^*
ACTUAL	π_1	**100**	0.0	0.0
	π_2	0.0	**100**	0.0
	π_3	0.0	0.0	**100**

SNR = 15 dB

INDEX: 100		SELECTED π_1^*	π_2^*	π_3^*
ACTUAL	π_1	**100**	0.0	0.0
	π_2	0.0	**100**	0.0
	π_3	0.0	0.0	**100**

SNR = 10 dB

INDEX: 100		SELECTED π_1^*	π_2^*	π_3^*
ACTUAL	π_1	**100**	0.0	0.0
	π_2	0.0	**100**	0.0
	π_3	0.0	0.0	**100**

SNR = 5 dB

INDEX: 100		SELECTED π_1^*	π_2^*	π_3^*
ACTUAL	π_1	**100**	0.0	0.0
	π_2	0.0	**100**	0.0
	π_3	0.0	0.0	**100**

SNR = 0 dB

INDEX: 99.4		SELECTED π_1^*	π_2^*	π_3^*
ACTUAL	π_1	**99.6**	0.1	0.2
	π_2	0.6	**99.0**	0.4
	π_3	0.2	0.2	**99.6**

SNR = -5 dB

INDEX: 96.9		SELECTED π_1^*	π_2^*	π_3^*
ACTUAL	π_1	**96.5**	1.4	2.1
	π_2	1.3	**97.5**	1.2
	π_3	2.1	1.0	**96.8**

SNR = -10 dB

INDEX: 93.8		SELECTED π_1^*	π_2^*	π_3^*
ACTUAL	π_1	**94.4**	2.8	2.8
	π_2	4.3	**93.3**	2.4
	π_3	4.0	2.2	**93.7**

SNR = -15 dB

INDEX: 96.4		SELECTED π_1^*	π_2^*	π_3^*
ACTUAL	π_1	**98.0**	1.2	0.8
	π_2	1.7	**96.2**	2.1
	π_3	2.5	2.5	**95.0**

SNR = -20 dB

Table B-3. Confusion Matrices for Simulated Feature Trials (Three-Class, Fifty-Features): Statistical Classifier. (see App B cover page for table description)

INDEX: 96.1		SELECTED π_1*	SELECTED π_2*	SELECTED π_3*
ACTUAL	π_1	**98.3**	0.3	0.5
	π_2	0.1	**94.5**	5.5
	π_3	0.0	4.4	**95.5**

SNR = 20 dB

INDEX: 95.2		SELECTED π_1*	SELECTED π_2*	SELECTED π_3*
ACTUAL	π_1	**96.0**	0.3	3.6
	π_2	0.2	**95.7**	2.2
	π_3	4.0	2.3	**93.7**

SNR = 15 dB

INDEX: 80.9		SELECTED π_1*	SELECTED π_2*	SELECTED π_3*
ACTUAL	π_1	**71.8**	2.8	23.4
	π_2	1.0	**77.4**	21.2
	π_3	2.6	3.9	**93.5**

SNR = 10 dB

INDEX: 73.5		SELECTED π_1*	SELECTED π_2*	SELECTED π_3*
ACTUAL	π_1	**78.7**	2.6	12.8
	π_2	2.6	**64.8**	28.2
	π_3	7.4	15.2	**77.0**

SNR = 5 dB

INDEX: 55.1		SELECTED π_1*	SELECTED π_2*	SELECTED π_3*
ACTUAL	π_1	**53.4**	8.8	33.8
	π_2	9.8	**52.3**	30.5
	π_3	20.5	16.8	**59.8**

SNR = 0 dB

INDEX: 35.5		SELECTED π_1*	SELECTED π_2*	SELECTED π_3*
ACTUAL	π_1	**31.6**	12.1	41.6
	π_2	16.3	**28.1**	43.7
	π_3	20.2	22.1	**46.8**

SNR = -5 dB

INDEX: 31.9		SELECTED π_1*	SELECTED π_2*	SELECTED π_3*
ACTUAL	π_1	**26.5**	15.7	42.6
	π_2	19.2	**23.5**	41.1
	π_3	23.4	18.4	**45.8**

SNR = -10 dB

INDEX: 28.4		SELECTED π_1*	SELECTED π_2*	SELECTED π_3*
ACTUAL	π_1	**16.9**	11.8	53.2
	π_2	14.0	**14.0**	54.9
	π_3	16.1	11.7	**54.3**

SNR = -15 dB

INDEX: 28.9		SELECTED π_1*	SELECTED π_2*	SELECTED π_3*
ACTUAL	π_1	**20.4**	18.7	44.8
	π_2	20.2	**17.9**	47.4
	π_3	19.7	18.6	**48.4**

SNR = -20 dB

Table B-4. Confusion Matrices for Simulated Feature Trials (Three-Class, Three-Features): Perceptron. (see App B cover page for table description)

INDEX: 99.9	SELECTED		
ACTUAL	π_1*	π_2*	π_3*
π_1	**99.9**	0.0	0.1
π_2	0.0	**100**	0.0
π_3	0.0	0.1	**99.9**

SNR = 20 dB

INDEX: 99.8	SELECTED		
ACTUAL	π_1*	π_2*	π_3*
π_1	**99.9**	0.0	0.1
π_2	0.0	**99.7**	0.1
π_3	0.0	0.0	**99.9**

SNR = 15 dB

INDEX: 99.6	SELECTED		
ACTUAL	π_1*	π_2*	π_3*
π_1	**99.8**	0.0	0.1
π_2	0.0	**99.7**	0.2
π_3	0.5	0.1	**99.4**

SNR = 10 dB

INDEX: 96.7	SELECTED		
ACTUAL	π_1*	π_2*	π_3*
π_1	**96.1**	0.2	2.4
π_2	0.3	**95.5**	2.7
π_3	0.8	0.7	**98.5**

SNR = 5 dB

INDEX: 84.0	SELECTED		
ACTUAL	π_1*	π_2*	π_3*
π_1	**80.7**	2.8	12.5
π_2	2.0	**82.5**	10.3
π_3	4.1	7.0	**88.8**

SNR = 0 dB

INDEX: 52.7	SELECTED		
ACTUAL	π_1*	π_2*	π_3*
π_1	**45.5**	6.6	36.2
π_2	7.3	**46.9**	32.1
π_3	15.1	15.0	**65.6**

SNR = -5 dB

INDEX: 38.7	SELECTED		
ACTUAL	π_1*	π_2*	π_3*
π_1	**30.0**	14.6	39.4
π_2	13.3	**31.5**	40.6
π_3	15.0	20.2	**54.7**

SNR = -10 dB

INDEX: 32.1	SELECTED		
ACTUAL	π_1*	π_2*	π_3*
π_1	**21.9**	10.8	49.3
π_2	14.4	**21.1**	48.1
π_3	15.7	15.0	**53.3**

SNR = -15 dB

INDEX: 28.3	SELECTED		
ACTUAL	π_1*	π_2*	π_3*
π_1	**16.5**	14.0	50.6
π_2	12.4	**15.4**	51.8
π_3	14.7	14.5	**52.9**

SNR = -20 dB

Table B-5. Confusion Matrices for Simulated Feature Trials (Three-Class, Ten-Features): Perceptron. (see App B cover page for table description)

INDEX: 99.9	SELECTED π_1^*	π_2^*	π_3^*
ACTUAL π_1	**99.8**	0.0	0.1
π_2	0.0	**100**	0.0
π_3	0.0	0.0	**100**

SNR = 20 dB

INDEX: 100	SELECTED π_1^*	π_2^*	π_3^*
ACTUAL π_1	**99.9**	0.0	0.0
π_2	0.0	**100**	0.0
π_3	0.0	0.1	**99.9**

SNR = 15 dB

INDEX: 99.8	SELECTED π_1^*	π_2^*	π_3^*
ACTUAL π_1	**99.7**	0.0	0.1
π_2	0.0	**99.9**	0.1
π_3	0.1	0.3	**99.7**

SNR = 10 dB

INDEX: 99.6	SELECTED π_1^*	π_2^*	π_3^*
ACTUAL π_1	**99.7**	0.0	0.2
π_2	0.0	**99.6**	0.2
π_3	0.2	0.2	**99.6**

SNR = 5 dB

INDEX: 99.3	SELECTED π_1^*	π_2^*	π_3^*
ACTUAL π_1	**99.4**	0.0	0.3
π_2	0.0	**99.3**	0.5
π_3	0.2	0.5	**99.3**

SNR = 0 dB

INDEX: 92.5	SELECTED π_1^*	π_2^*	π_3^*
ACTUAL π_1	**90.4**	0.6	2.9
π_2	0.4	**93.3**	3.5
π_3	2.4	3.6	**94.0**

SNR = -5 dB

INDEX: 71.1	SELECTED π_1^*	π_2^*	π_3^*
ACTUAL π_1	**71.6**	5.0	16.6
π_2	6.2	**70.6**	15.5
π_3	12.4	15.8	**71.1**

SNR = -10 dB

INDEX: 47.2	SELECTED π_1^*	π_2^*	π_3^*
ACTUAL π_1	**44.2**	16.3	30.6
π_2	17.9	**40.9**	30.3
π_3	20.8	18.6	**56.4**

SNR = -15 dB

INDEX: 34.8	SELECTED π_1^*	π_2^*	π_3^*
ACTUAL π_1	**28.5**	16.2	38.5
π_2	14.9	**28.9**	39.9
π_3	19.0	20.0	**47.0**

SNR = -20 dB

Table B-6. Confusion Matrices for Simulated Feature Trials (Three-Class, Fifty-Features): Perceptron. (see App B cover page for table description)

INDEX: 99.6	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**99.8**	0.2	0.0
π_2	0.6	**99.3**	0.2
π_3	0.2	0.0	**99.7**

SNR = 20 dB

INDEX: 96.8	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**98.3**	0.0	1.6
π_2	0.0	**96.0**	4.0
π_3	0.6	3.2	**96.2**

SNR = 15 dB

INDEX: 95.0	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**89.5**	1.2	9.3
π_2	0.7	**98.8**	0.5
π_3	3.2	0.2	**96.5**

SNR = 10 dB

INDEX: 89.0	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**86.8**	5.6	7.6
π_2	1.0	**96.4**	2.7
π_3	6.8	9.4	**83.7**

SNR = 5 dB

INDEX: 65.3	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**58.0**	24.0	18.1
π_2	28.1	**64.1**	7.8
π_3	16.3	9.7	**74.0**

SNR = 0 dB

INDEX: 54.5	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**51.4**	21.6	27.0
π_2	22.4	**56.4**	21.2
π_3	18.5	25.7	**55.9**

SNR = -5 dB

INDEX: 48.2	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**54.7**	22.8	22.6
π_2	25.4	**50.8**	23.8
π_3	27.4	33.6	**39.0**

SNR = -10 dB

INDEX: 45.0	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**54.6**	17.3	28.0
π_2	36.1	**24.1**	39.9
π_3	31.5	12.3	**56.3**

SNR = -15 dB

INDEX: 37.8	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**32.2**	37.3	30.6
π_2	33.5	**45.3**	21.1
π_3	31.3	32.8	**35.9**

SNR = -20 dB

Table B-7. Confusion Matrices for Simulated Feature Trials (Three-Class, Three-Features): MSNN. (see App B cover page for table description)

INDEX: 99.8		SELECTED		
		π_1*	π_2*	π_3*
ACTUAL	π_1	**99.7**	0.0	0.3
	π_2	0.0	**100**	0.0
	π_3	0.1	0.0	**99.9**

SNR = 20 dB

INDEX: 99.9		SELECTED		
		π_1*	π_2*	π_3*
ACTUAL	π_1	**100**	0.0	0.0
	π_2	0.1	**99.8**	0.0
	π_3	0.0	0.0	**100**

SNR = 15 dB

INDEX: 99.4		SELECTED		
		π_1*	π_2*	π_3*
ACTUAL	π_1	**99.5**	0.3	0.3
	π_2	0.6	**99.3**	0.1
	π_3	0.2	0.4	**99.4**

SNR = 10 dB

INDEX: 97.8		SELECTED		
		π_1*	π_2*	π_3*
ACTUAL	π_1	**97.3**	0.4	2.4
	π_2	0.8	**98.3**	0.9
	π_3	1.2	0.9	**97.9**

SNR = 5 dB

INDEX: 90.4		SELECTED		
		π_1*	π_2*	π_3*
ACTUAL	π_1	**93.8**	3.5	2.7
	π_2	4.0	**89.9**	6.1
	π_3	6.1	6.3	**87.5**

SNR = 0 dB

INDEX: 68.2		SELECTED		
		π_1*	π_2*	π_3*
ACTUAL	π_1	**59.4**	22.6	18.0
	π_2	15.9	**70.9**	13.2
	π_3	13.4	12.4	**74.2**

SNR = -5 dB

INDEX: 53.6		SELECTED		
		π_1*	π_2*	π_3*
ACTUAL	π_1	**46.0**	28.2	25.8
	π_2	21.2	**56.7**	22.1
	π_3	18.4	23.4	**58.2**

SNR = -10 dB

INDEX: 44.4		SELECTED		
		π_1*	π_2*	π_3*
ACTUAL	π_1	**44.9**	29.1	26.0
	π_2	25.8	**45.4**	28.8
	π_3	27.9	29.2	**42.8**

SNR = -15 dB

INDEX: 36.7		SELECTED		
		π_1*	π_2*	π_3*
ACTUAL	π_1	**35.8**	32.9	31.3
	π_2	29.0	**38.7**	32.3
	π_3	31.9	32.4	**35.7**

SNR = -20 dB

Table B-8. Confusion Matrices for Simulated Feature Trials (Three-Class, Ten-Features): MSNN. (see App B cover page for table description)

INDEX: 100		SELECTED		
		π_1^*	π_2^*	π_3^*
ACTUAL	π_1	**100**	0.0	0.0
	π_2	0.0	**100**	0.0
	π_3	0.0	0.0	**100**

SNR = 20 dB

INDEX: 99.9		SELECTED		
		π_1^*	π_2^*	π_3^*
ACTUAL	π_1	**99.8**	0.0	0.2
	π_2	0.0	**100**	0.0
	π_3	0.0	0.0	**100**

SNR = 15 dB

INDEX: 99.8		SELECTED		
		π_1^*	π_2^*	π_3^*
ACTUAL	π_1	**99.5**	0.3	0.2
	π_2	0.0	**100**	0.0
	π_3	0.0	0.0	**100**

SNR = 10 dB

INDEX: 99.5		SELECTED		
		π_1^*	π_2^*	π_3^*
ACTUAL	π_1	**99.2**	0.2	0.5
	π_2	0.3	**99.6**	0.1
	π_3	0.2	0.1	**99.7**

SNR = 5 dB

INDEX: 98.8		SELECTED		
		π_1^*	π_2^*	π_3^*
ACTUAL	π_1	**99.1**	0.6	0.3
	π_2	1.0	**98.3**	0.7
	π_3	0.4	0.4	**99.1**

SNR = 0 dB

INDEX: 95.3		SELECTED		
		π_1^*	π_2^*	π_3^*
ACTUAL	π_1	**95.2**	2.5	2.3
	π_2	2.0	**96.1**	1.9
	π_3	3.3	2.2	**94.6**

SNR = -5 dB

INDEX: 75.3		SELECTED		
		π_1^*	π_2^*	π_3^*
ACTUAL	π_1	**76.7**	10.0	13.3
	π_2	10.4	**76.5**	13.1
	π_3	14.1	13.2	**72.7**

SNR = -10 dB

INDEX: 56.0		SELECTED		
		π_1^*	π_2^*	π_3^*
ACTUAL	π_1	**55.5**	22.5	22.0
	π_2	22.8	**55.5**	21.6
	π_3	21.0	22.1	**56.9**

SNR = -15 dB

INDEX: 39.4		SELECTED		
		π_1^*	π_2^*	π_3^*
ACTUAL	π_1	**41.6**	30.4	28.0
	π_2	34.5	**38.8**	26.8
	π_3	31.9	30.2	**37.9**

SNR = -20 dB

Table B-9. Confusion Matrices for Simulated Feature Trials (Three-Class, Fifty-Features): MSNN. (see App B cover page for table description)

INDEX: 89.4	SELECTED π₁*	π₂*	π₃*
ACTUAL π_1	**90.4**	8.1	1.6
π_2	5.6	**92.5**	1.9
π_3	7.1	7.7	**85.2**

SNR = 20 dB

INDEX: 83.9	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**82.9**	1.3	15.8
π_2	2.2	**77.4**	20.4
π_3	4.0	4.7	**91.3**

SNR = 15 dB

INDEX: 85.8	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**87.3**	1.0	11.7
π_2	4.7	**85.3**	10.1
π_3	14.9	0.4	**84.7**

SNR = 10 dB

INDEX: 80.4	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**83.9**	7.0	9.2
π_2	8.6	**83.0**	8.4
π_3	14.4	11.2	**74.4**

SNR = 5 dB

INDEX: 62.6	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**54.8**	18.9	26.4
π_2	32.1	**53.1**	14.9
π_3	14.2	5.9	**79.9**

SNR = 0 dB

INDEX: 52.2	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**47.2**	25.5	27.3
π_2	17.8	**57.2**	25.0
π_3	19.5	28.4	**52.2**

SNR = -5 dB

INDEX: 42.7	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**44.9**	30.9	24.2
π_2	29.9	**44.1**	26.0
π_3	28.3	32.7	**39.0**

SNR = -10 dB

INDEX: 41.8	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**44.7**	28.4	27.0
π_2	28.2	**37.3**	34.5
π_3	28.6	28.1	**43.3**

SNR = -15 dB

INDEX: 35.7	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**63.7**	17.8	18.5
π_2	60.6	**19.5**	20.0
π_3	58.3	17.6	**24.0**

SNR = -20 dB

Table B-10. Confusion Matrices for Simulated Feature Trials (Three-Class, Three-Features): MSNN Mod 1. (see App B cover page for table description)

INDEX: 96.0		SELECTED		
		π_1*	π_2*	π_3*
ACTUAL	π_1	**97.4**	2.2	0.5
	π_2	3.0	**96.2**	0.8
	π_3	2.6	3.0	**94.4**

SNR = 20 dB

INDEX: 96.3		SELECTED		
		π_1*	π_2*	π_3*
ACTUAL	π_1	**96.3**	1.5	2.3
	π_2	1.8	**97.7**	0.5
	π_3	2.8	2.4	**94.8**

SNR = 15 dB

INDEX: 97.4		SELECTED		
		π_1*	π_2*	π_3*
ACTUAL	π_1	**96.6**	0.8	2.6
	π_2	0.5	**97.4**	2.2
	π_3	0.7	1.0	**98.3**

SNR = 10 dB

INDEX: 94.7		SELECTED		
		π_1*	π_2*	π_3*
ACTUAL	π_1	**96.2**	0.7	3.1
	π_2	5.2	**92.6**	2.2
	π_3	3.6	1.1	**95.3**

SNR = 5 dB

INDEX: 88.6		SELECTED		
		π_1*	π_2*	π_3*
ACTUAL	π_1	**94.1**	2.6	3.4
	π_2	7.0	**87.1**	5.9
	π_3	7.2	8.4	**84.4**

SNR = 0 dB

INDEX: 67.4		SELECTED		
		π_1*	π_2*	π_3*
ACTUAL	π_1	**57.8**	20.8	21.4
	π_2	15.9	**68.7**	15.4
	π_3	11.5	12.7	**75.8**

SNR = -5 dB

INDEX: 55.7		SELECTED		
		π_1*	π_2*	π_3*
ACTUAL	π_1	**39.6**	30.0	30.4
	π_2	15.1	**57.9**	27.0
	π_3	10.5	20.0	**69.5**

SNR = -10 dB

INDEX: 47.5		SELECTED		
		π_1*	π_2*	π_3*
ACTUAL	π_1	**46.3**	24.2	29.4
	π_2	25.2	**44.1**	30.7
	π_3	24.6	23.4	**52.0**

SNR = -15 dB

INDEX: 43.1		SELECTED		
		π_1*	π_2*	π_3*
ACTUAL	π_1	**35.7**	29.2	35.0
	π_2	23.4	**45.4**	31.2
	π_3	22.6	29.0	**48.3**

SNR = -20 dB

Table B-11. Confusion Matrices for Simulated Feature Trials (Three-Class, Ten-Features): MSNN Mod 1. (see App B cover page for table description)

INDEX: 100	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**100**	0.0	0.0
π_2	0.0	**100**	0.0
π_3	0.0	0.0	**100**

SNR = 20 dB

INDEX: 100	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**100**	0.0	0.0
π_2	0.0	**100**	0.0
π_3	0.0	0.0	**100**

SNR = 15 dB

INDEX: 100	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**99.9**	0.0	0.0
π_2	0.0	**100**	0.0
π_3	0.0	0.0	**100**

SNR = 10 dB

INDEX: 100	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**100**	0.0	0.0
π_2	0.0	**100**	0.0
π_3	0.1	0.0	**99.9**

SNR = 5 dB

INDEX: 99.7	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**99.8**	0.0	0.2
π_2	0.2	**99.8**	0.0
π_3	0.4	0.1	**99.5**

SNR = 0 dB

INDEX: 96.6	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**94.8**	2.1	3.1
π_2	0.6	**97.5**	1.8
π_3	1.1	1.5	**97.4**

SNR = -5 dB

INDEX: 66.2	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**80.7**	10.1	9.1
π_2	14.7	**69.5**	15.9
π_3	31.1	20.5	**48.4**

SNR = -10 dB

INDEX: 53.4	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**55.8**	20.3	23.9
π_2	33.6	**42.9**	23.6
π_3	24.2	14.3	**61.4**

SNR = -15 dB

INDEX: 46.9	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**37.1**	41.0	21.9
π_2	20.8	**59.5**	19.7
π_3	20.2	35.8	**44.0**

SNR = -20 dB

Table B-12. Confusion Matrices for Simulated Feature Trials (Three-Class, Fifty-Features): MSNN Mod 1. (see App B cover page for table description)

INDEX: 99.5	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**98.9**	0.1	1.0
π_2	0.1	**99.9**	0.0
π_3	0.3	0.1	**99.7**

SNR = 20 dB

INDEX: 96.8	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**98.7**	0.0	1.3
π_2	0.0	**96.2**	3.8
π_3	0.8	3.7	**95.5**

SNR = 15 dB

INDEX: 93.2	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**89.0**	0.8	10.3
π_2	0.7	**98.7**	0.6
π_3	6.7	1.5	**91.9**

SNR = 10 dB

INDEX: 87.1	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**82.8**	5.3	11.8
π_2	1.7	**95.7**	2.5
π_3	7.4	9.8	**82.8**

SNR = 5 dB

INDEX: 63.2	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**55.5**	23.2	21.3
π_2	31.1	**59.2**	9.7
π_3	15.4	9.8	**74.8**

SNR = 0 dB

INDEX: 54.7	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**43.6**	28.4	28.1
π_2	16.5	**61.6**	21.9
π_3	13.7	27.3	**59.0**

SNR = -5 dB

INDEX: 48.0	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**56.6**	19.3	24.1
π_2	28.2	**44.8**	27.0
π_3	28.9	28.7	**42.4**

SNR = -10 dB

INDEX: 42.2	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**45.7**	24.8	29.5
π_2	31.5	**31.7**	36.8
π_3	28.5	22.2	**49.3**

SNR = -15 dB

INDEX: 36.5	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**32.7**	38.0	29.3
π_2	32.9	**43.2**	23.9
π_3	31.5	35.0	**33.6**

SNR = -20 dB

Table B-13. Confusion Matrices for Simulated Feature Trials (Three-Class, Three-Features): MSNN Mod 2. (see App B cover page for table description)

INDEX: 100	SELECTED		
ACTUAL	π_1*	π_2*	π_3*
π_1	**100**	0.0	0.0
π_2	0.0	**100**	0.0
π_3	0.0	0.0	**100**

SNR = 20 dB

INDEX: 99.9	SELECTED		
ACTUAL	π_1*	π_2*	π_3*
π_1	**99.9**	0.0	0.1
π_2	0.0	**99.8**	0.2
π_3	0.1	0.0	**99.9**

SNR = 15 dB

INDEX: 99.4	SELECTED		
ACTUAL	π_1*	π_2*	π_3*
π_1	**99.3**	0.4	0.3
π_2	0.5	**99.4**	0.1
π_3	0.3	0.3	**99.4**

SNR = 10 dB

INDEX: 98.2	SELECTED		
ACTUAL	π_1*	π_2*	π_3*
π_1	**98.3**	0.9	0.7
π_2	0.6	**98.4**	1.0
π_3	1.1	1.0	**97.8**

SNR = 5 dB

INDEX: 90.7	SELECTED		
ACTUAL	π_1*	π_2*	π_3*
π_1	**93.8**	3.4	2.8
π_2	3.2	**90.5**	6.3
π_3	6.2	5.9	**87.9**

SNR = 0 dB

INDEX: 67.9	SELECTED		
ACTUAL	π_1*	π_2*	π_3*
π_1	**61.8**	20.2	18.0
π_2	17.2	**68.8**	13.9
π_3	14.9	12.2	**73.0**

SNR = -5 dB

INDEX: 53.4	SELECTED		
ACTUAL	π_1*	π_2*	π_3*
π_1	**45.7**	28.6	25.7
π_2	20.9	**56.5**	22.6
π_3	18.4	23.4	**58.1**

SNR = -10 dB

INDEX: 42.0	SELECTED		
ACTUAL	π_1*	π_2*	π_3*
π_1	**45.2**	30.5	24.3
π_2	31.3	**40.9**	27.8
π_3	29.8	30.4	**39.8**

SNR = -15 dB

INDEX: 35.0	SELECTED		
ACTUAL	π_1*	π_2*	π_3*
π_1	**34.3**	31.8	33.8
π_2	30.7	**36.4**	32.9
π_3	33.3	32.3	**34.4**

SNR = -20 dB

Table B-14. Confusion Matrices for Simulated Feature Trials (Three-Class, Ten-Features): MSNN Mod 2. (see App B cover page for table description)

INDEX: 100		SELECTED π_1*	π_2*	π_3*
ACTUAL	π_1	**100**	0.0	0.0
	π_2	0.0	**100**	0.0
	π_3	0.0	0.0	**100**

SNR = 20 dB

INDEX: 99.9		SELECTED π_1*	π_2*	π_3*
ACTUAL	π_1	**99.8**	0.0	0.2
	π_2	0.0	**100**	0.0
	π_3	0.0	0.0	**100**

SNR = 15 dB

INDEX: 99.7		SELECTED π_1*	π_2*	π_3*
ACTUAL	π_1	**99.7**	0.3	0.0
	π_2	0.2	**99.7**	0.0
	π_3	0.2	0.0	**99.8**

SNR = 10 dB

INDEX: 99.2		SELECTED π_1*	π_2*	π_3*
ACTUAL	π_1	**99.5**	0.3	0.1
	π_2	0.1	**98.7**	1.2
	π_3	0.0	0.4	**99.5**

SNR = 5 dB

INDEX: 99.0		SELECTED π_1*	π_2*	π_3*
ACTUAL	π_1	**98.4**	1.1	0.4
	π_2	0.5	**99.2**	0.3
	π_3	0.4	0.3	**99.3**

SNR = 0 dB

INDEX: 95.6		SELECTED π_1*	π_2*	π_3*
ACTUAL	π_1	**95.5**	1.8	2.6
	π_2	1.7	**96.3**	2.0
	π_3	2.4	2.7	**94.9**

SNR = -5 dB

INDEX: 71.9		SELECTED π_1*	π_2*	π_3*
ACTUAL	π_1	**73.2**	12.0	14.8
	π_2	11.6	**74.2**	14.2
	π_3	15.5	16.2	**68.2**

SNR = -10 dB

INDEX: 47.4		SELECTED π_1*	π_2*	π_3*
ACTUAL	π_1	**43.9**	27.8	28.3
	π_2	24.5	**49.1**	26.4
	π_3	22.6	27.2	**50.2**

SNR = -15 dB

INDEX: 36.3		SELECTED π_1*	π_2*	π_3*
ACTUAL	π_1	**36.9**	35.2	28.0
	π_2	32.9	**39.3**	27.8
	π_3	32.8	34.6	**32.7**

SNR = -20 dB

Table B-15. Confusion Matrices for Simulated Feature Trials (Three-Class, Fifty-Features): MSNN Mod 2. (see App B cover page for table description)

INDEX: 98.5	SELECTED π_1^*	π_2^*	π_3^*
ACTUAL π_1	**99.6**	0.3	0.0
π_2	0.6	**98.7**	0.7
π_3	0.2	2.5	**97.3**

SNR = 20 dB

INDEX: 96.0	SELECTED π_1^*	π_2^*	π_3^*
ACTUAL π_1	**97.1**	0.0	2.9
π_2	0.0	**96.5**	3.5
π_3	0.7	4.9	**94.4**

SNR = 15 dB

INDEX: 88.0	SELECTED π_1^*	π_2^*	π_3^*
ACTUAL π_1	**80.2**	7.3	12.5
π_2	1.0	**97.7**	1.3
π_3	7.4	6.4	**86.1**

SNR = 10 dB

INDEX: 79.8	SELECTED π_1^*	π_2^*	π_3^*
ACTUAL π_1	**74.1**	11.9	14.0
π_2	8.8	**87.4**	3.9
π_3	8.4	13.7	**77.9**

SNR = 5 dB

INDEX: 61.4	SELECTED π_1^*	π_2^*	π_3^*
ACTUAL π_1	**51.6**	21.7	26.7
π_2	28.0	**55.9**	16.1
π_3	13.7	9.4	**76.9**

SNR = 0 dB

INDEX: 49.4	SELECTED π_1^*	π_2^*	π_3^*
ACTUAL π_1	**46.2**	25.4	28.4
π_2	23.2	**52.2**	24.6
π_3	20.6	29.5	**49.9**

SNR = -5 dB

INDEX: 43.6	SELECTED π_1^*	π_2^*	π_3^*
ACTUAL π_1	**48.0**	28.7	23.3
π_2	27.3	**45.8**	26.9
π_3	28.3	34.7	**37.0**

SNR = -10 dB

INDEX: 40.2	SELECTED π_1^*	π_2^*	π_3^*
ACTUAL π_1	**40.0**	28.4	31.6
π_2	29.7	**34.5**	35.8
π_3	30.8	23.3	**46.0**

SNR = -15 dB

INDEX: 36.0	SELECTED π_1^*	π_2^*	π_3^*
ACTUAL π_1	**29.7**	39.9	30.3
π_2	31.5	**45.2**	23.3
π_3	28.8	38.1	**33.1**

SNR = -20 dB

Table B-16. Confusion Matrices for Simulated Feature Trials (Three-Class, Three-Features): MSNN Mod 3. (see App B cover page for table description)

INDEX: 99.2	SELECTED		
	π_1*	π_2*	π_3*
ACTUAL π_1	**98.1**	0.0	1.9
ACTUAL π_2	0.3	**99.7**	0.0
ACTUAL π_3	0.1	0.0	**99.9**

SNR = 20 dB

INDEX: 99.2	SELECTED		
	π_1*	π_2*	π_3*
ACTUAL π_1	**100**	0.0	0.0
ACTUAL π_2	1.1	**98.9**	0.0
ACTUAL π_3	0.6	0.6	**98.8**

SNR = 15 dB

INDEX: 97.1	SELECTED		
	π_1*	π_2*	π_3*
ACTUAL π_1	**97.7**	2.1	0.2
ACTUAL π_2	1.6	**95.9**	2.5
ACTUAL π_3	1.1	1.2	**97.7**

SNR = 10 dB

INDEX: 90.5	SELECTED		
	π_1*	π_2*	π_3*
ACTUAL π_1	**92.6**	2.2	5.2
ACTUAL π_2	4.8	**93.1**	2.1
ACTUAL π_3	8.6	5.6	**85.8**

SNR = 5 dB

INDEX: 79.0	SELECTED		
	π_1*	π_2*	π_3*
ACTUAL π_1	**84.5**	6.7	8.8
ACTUAL π_2	9.4	**79.9**	10.6
ACTUAL π_3	15.4	12.1	**72.4**

SNR = 0 dB

INDEX: 59.5	SELECTED		
	π_1*	π_2*	π_3*
ACTUAL π_1	**51.4**	20.5	28.1
ACTUAL π_2	21.2	**57.4**	21.4
ACTUAL π_3	17.5	12.6	**69.8**

SNR = -5 dB

INDEX: 46.0	SELECTED		
	π_1*	π_2*	π_3*
ACTUAL π_1	**39.5**	33.1	27.3
ACTUAL π_2	24.1	**50.4**	25.4
ACTUAL π_3	24.5	27.3	**48.2**

SNR = -10 dB

INDEX: 39.2	SELECTED		
	π_1*	π_2*	π_3*
ACTUAL π_1	**39.4**	36.6	23.9
ACTUAL π_2	29.8	**43.5**	26.7
ACTUAL π_3	29.3	35.9	**34.8**

SNR = -15 dB

INDEX: 34.7	SELECTED		
	π_1*	π_2*	π_3*
ACTUAL π_1	**29.1**	38.6	32.2
ACTUAL π_2	26.9	**41.0**	32.1
ACTUAL π_3	28.4	37.6	**34.1**

SNR = -20 dB

Table B-17. Confusion Matrices for Simulated Feature Trials (Three-Class, Ten-Features): MSNN Mod 3. (see App B cover page for table description)

INDEX: 100	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**100**	0.0	0.0
π_2	0.0	**100**	0.0
π_3	0.0	0.0	**100**

SNR = 20 dB

INDEX: 98.7	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**99.8**	0.0	0.2
π_2	3.2	**96.2**	0.6
π_3	0.0	0.0	**100**

SNR = 15 dB

INDEX: 98.4	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**97.2**	0.7	2.2
π_2	1.4	**98.1**	0.5
π_3	0.0	0.1	**99.9**

SNR = 10 dB

INDEX: 97.7	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**96.9**	2.1	1.0
π_2	1.7	**97.2**	1.1
π_3	0.6	0.2	**99.1**

SNR = 5 dB

INDEX: 91.5	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**89.4**	4.4	6.3
π_2	4.2	**93.5**	2.3
π_3	4.9	3.6	**91.5**

SNR = 0 dB

INDEX: 78.4	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**79.4**	7.2	13.4
π_2	9.9	**77.0**	13.1
π_3	13.0	8.3	**78.6**

SNR = -5 dB

INDEX: 56.7	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**60.1**	18.6	21.2
π_2	22.2	**59.5**	18.3
π_3	25.9	23.6	**50.5**

SNR = -10 dB

INDEX: 43.8	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**44.7**	27.3	28.0
π_2	30.3	**42.0**	27.7
π_3	28.3	27.1	**44.6**

SNR = -15 dB

INDEX: 36.1	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**35.3**	37.0	27.6
π_2	32.4	**41.4**	26.2
π_3	32.4	36.1	**31.5**

SNR = -20 dB

Table B-18. Confusion Matrices for Simulated Feature Trials (Three-Class, Fifty-Features): MSNN Mod 3. (see App B cover page for table description)

INDEX: 75.2	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**61.8**	38.1	0.0
ACTUAL π_2	35.9	**63.8**	0.4
ACTUAL π_3	0.0	0.0	**100**

SNR = 20 dB

INDEX: 71.3	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**57.4**	42.5	0.1
ACTUAL π_2	43.1	**56.5**	0.4
ACTUAL π_3	0.0	0.0	**100**

SNR = 15 dB

INDEX: 67.9	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**54.2**	44.8	1.0
ACTUAL π_2	47.5	**49.8**	2.7
ACTUAL π_3	0.1	0.2	**99.7**

SNR = 10 dB

INDEX: 60.2	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**44.0**	40.6	15.4
ACTUAL π_2	41.0	**42.9**	16.1
ACTUAL π_3	4.1	2.3	**93.6**

SNR = 5 dB

INDEX: 44.6	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**35.9**	35.7	28.4
ACTUAL π_2	34.8	**35.2**	30.0
ACTUAL π_3	18.6	18.7	**62.6**

SNR = 0 dB

INDEX: 35.9	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**34.0**	30.6	35.4
ACTUAL π_2	33.2	**33.2**	33.7
ACTUAL π_3	30.9	28.6	**40.5**

SNR = -5 dB

INDEX: 33.3	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**33.5**	30.0	36.5
ACTUAL π_2	34.3	**29.8**	35.8
ACTUAL π_3	33.4	30.0	**36.6**

R x 1

INDEX: 33.6	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**31.7**	32.7	35.6
ACTUAL π_2	32.1	**32.3**	35.6
ACTUAL π_3	31.1	32.2	**36.7**

SNR = -15 dB

INDEX: 33.3	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**37.4**	30.9	31.7
ACTUAL π_2	36.7	**31.6**	31.7
ACTUAL π_3	38.3	30.8	**30.9**

SNR = -20 dB

INDEX: 33.3	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**100**	0.0	0.0
ACTUAL π_2	100	**0.0**	0.0
ACTUAL π_3	100	0.0	**0.0**

No Noise

Table B-19. Confusion Matrices for Simulated Modulated Signals (Three-Class, Fifty-One Features): Statistical Classifier. (see App B cover page for table description)

INDEX: 79.4	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**68.0**	32.0	0.0
π_2	29.9	**70.1**	0.0
π_3	0.0	0.0	**100**

SNR = 20 dB

INDEX: 74.7	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**60.0**	40.0	0.0
π_2	35.6	**64.3**	0.0
π_3	0.0	0.0	**100**

SNR = 15 dB

INDEX: 70.9	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**56.4**	43.6	0.1
π_2	43.3	**56.5**	0.2
π_3	0.1	0.2	**99.8**

SNR = 10 dB

INDEX: 65.7	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**51.2**	45.5	3.3
π_2	44.5	**50.6**	4.9
π_3	1.8	2.9	**95.3**

SNR = 5 dB

INDEX: 50.4	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**40.3**	42.0	17.7
π_2	38.7	**43.2**	18.1
π_3	13.8	18.4	**67.8**

SNR = 0 dB

INDEX: 37.0	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**34.5**	34.5	31.0
π_2	35.4	**34.1**	30.6
π_3	29.5	28.0	**42.4**

SNR = -5 dB

INDEX: 33.6	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**36.2**	32.3	31.5
π_2	36.4	**30.9**	32.7
π_3	34.0	32.2	**33.7**

SNR = -10 dB

INDEX: 33.3	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**33.7**	32.0	34.4
π_2	34.0	**32.1**	33.9
π_3	33.0	32.8	**34.2**

SNR = -15 dB

INDEX: 33.3	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**34.5**	33.2	32.2
π_2	34.8	**34.1**	31.1
π_3	34.6	34.0	**31.4**

SNR = -20 dB

INDEX: 33.3	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**40.0**	0.0	60.0
π_2	40.0	**0.0**	60.0
π_3	40.0	0.0	**60.0**

No Noise

Table B-20. Confusion Matrices for Simulated Modulated Signals (Three-Class, Twenty-Six Features): Statistical Classifier. (see App B cover page for table description)

INDEX: 78.8	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**68.0**	32.0	0.0
π_2	31.7	**68.3**	0.0
π_3	0.0	0.0	**100**

SNR = 20 dB

INDEX: 75.4	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**65.5**	34.3	0.2
π_2	39.2	**60.7**	0.1
π_3	0.0	0.0	**100**

SNR = 15 dB

INDEX: 73.0	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**61.9**	37.5	0.6
π_2	41.7	**57.7**	0.6
π_3	0.4	0.1	**99.4**

SNR = 10 dB

INDEX: 65.9	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**51.4**	45.8	2.8
π_2	42.7	**53.6**	3.7
π_3	3.0	4.3	**92.7**

SNR = 5 dB

INDEX: 51.7	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**43.0**	42.6	14.5
π_2	42.7	**43.5**	13.8
π_3	15.1	16.2	**68.7**

SNR = 0 dB

INDEX: 38.3	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**37.1**	37.0	26.0
π_2	37.5	**35.9**	26.6
π_3	27.8	30.3	**41.9**

SNR = -5 dB

INDEX: 34.7	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**34.2**	31.6	34.3
π_2	33.0	**32.3**	34.7
π_3	31.3	31.2	**37.5**

SNR = -10 dB

INDEX: 33.5	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**31.6**	33.7	34.7
π_2	32.0	**33.8**	34.2
π_3	30.8	34.0	**35.2**

SNR = -15 dB

INDEX: 33.1	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**31.6**	34.7	33.7
π_2	31.9	**35.9**	32.2
π_3	32.3	35.9	**31.7**

SNR = -20 dB

INDEX: 57.0	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**45.7**	54.3	0.0
π_2	54.7	**45.3**	0.0
π_3	0.0	20.0	**80.0**

No Noise

Table B-21. Confusion Matrices for Simulated Modulated Signals (Three-Class, Eleven-Features): Statistical Classifier. (see App B cover page for table description)

INDEX: 78.7		SELECTED π_1^*	π_2^*	π_3^*
ACTUAL	π_1	**65.6**	13.6	4.9
	π_2	11.0	**71.4**	7.9
	π_3	0.4	0.6	**99.0**

SNR = 20 dB

INDEX: 71.7		SELECTED π_1^*	π_2^*	π_3^*
ACTUAL	π_1	**63.3**	16.0	9.0
	π_2	23.8	**53.3**	10.8
	π_3	0.5	1.2	**98.3**

SNR = 15 dB

INDEX: 63.8		SELECTED π_1^*	π_2^*	π_3^*
ACTUAL	π_1	**47.9**	26.4	14.4
	π_2	27.6	**47.0**	15.2
	π_3	1.1	2.3	**96.6**

SNR = 10 dB

INDEX: 50.7		SELECTED π_1^*	π_2^*	π_3^*
ACTUAL	π_1	**27.7**	25.6	35.3
	π_2	22.9	**31.0**	35.2
	π_3	2.9	3.6	**93.4**

SNR = 5 dB

INDEX: 40.6		SELECTED π_1^*	π_2^*	π_3^*
ACTUAL	π_1	**24.0**	23.6	31.3
	π_2	22.9	**25.0**	31.7
	π_3	12.4	12.2	**72.8**

SNR = 0 dB

INDEX: 33.1		SELECTED π_1^*	π_2^*	π_3^*
ACTUAL	π_1	**25.0**	29.8	32.6
	π_2	24.5	**29.9**	32.8
	π_3	23.7	25.3	**44.2**

SNR = -5 dB

INDEX: 30.1		SELECTED π_1^*	π_2^*	π_3^*
ACTUAL	π_1	**27.7**	34.2	26.4
	π_2	28.1	**34.8**	25.9
	π_3	27.2	24.6	**27.9**

SNR = -10 dB

INDEX: 31.2		SELECTED π_1^*	π_2^*	π_3^*
ACTUAL	π_1	**25.0**	20.3	47.6
	π_2	24.1	**20.2**	48.9
	π_3	24.1	20.7	**48.5**

SNR = -15 dB

INDEX: 29.6		SELECTED π_1^*	π_2^*	π_3^*
ACTUAL	π_1	**15.1**	24.0	50.1
	π_2	15.6	**23.4**	50.0
	π_3	15.6	23.2	**50.4**

SNR = -20 dB

INDEX: 92.9		SELECTED π_1^*	π_2^*	π_3^*
ACTUAL	π_1	**91.4**	5.2	2.6
	π_2	7.0	**87.2**	4.6
	π_3	0.1	0.0	**99.9**

No Noise

Table B-22. Confusion Matrices for Simulated Modulated Signals (Three-Class, Fifty-One Features): Perceptron. (see App B cover page for table description)

INDEX: 74.2		SELECTED π_1*	π_2*	π_3*
ACTUAL	π_1	**55.4**	19.4	5.3
	π_2	11.7	**69.0**	5.9
	π_3	0.1	1.8	**98.1**

SNR = 20 dB

INDEX: 69.3		SELECTED π_1*	π_2*	π_3*
ACTUAL	π_1	**62.6**	16.1	9.8
	π_2	29.1	**46.4**	14.0
	π_3	0.2	1.1	**98.7**

SNR = 15 dB

INDEX: 61.4		SELECTED π_1*	π_2*	π_3*
ACTUAL	π_1	**24.0**	43.5	16.3
	π_2	8.1	**66.0**	14.4
	π_3	0.4	5.4	**94.2**

SNR = 10 dB

INDEX: 51.8		SELECTED π_1*	π_2*	π_3*
ACTUAL	π_1	**39.3**	20.5	30.9
	π_2	31.1	**26.3**	33.9
	π_3	4.1	5.9	**89.9**

SNR = 5 dB

INDEX: 40.0		SELECTED π_1*	π_2*	π_3*
ACTUAL	π_1	**28.5**	28.5	21.3
	π_2	26.5	**30.5**	22.4
	π_3	14.8	20.5	**61.1**

SNR = 0 dB

INDEX: 30.3		SELECTED π_1*	π_2*	π_3*
ACTUAL	π_1	**18.5**	19.1	43.3
	π_2	19.0	**19.2**	43.2
	π_3	18.0	16.2	**53.1**

SNR = -5 dB

INDEX: 28.6		SELECTED π_1*	π_2*	π_3*
ACTUAL	π_1	**18.4**	21.9	44.1
	π_2	18.5	**21.7**	44.5
	π_3	18.6	21.5	**45.7**

SNR = -10 dB

INDEX: 30.4		SELECTED π_1*	π_2*	π_3*
ACTUAL	π_1	**35.7**	24.9	30.4
	π_2	35.6	**25.2**	30.3
	π_3	36.0	25.0	**30.4**

SNR = -15 dB

INDEX: 30.4		SELECTED π_1*	π_2*	π_3*
ACTUAL	π_1	**22.8**	27.8	40.7
	π_2	23.0	**27.7**	40.3
	π_3	22.8	27.7	**40.6**

SNR = -20 dB

INDEX: 87.7		SELECTED π_1*	π_2*	π_3*
ACTUAL	π_1	**79.5**	10.0	4.5
	π_2	6.6	**83.9**	5.4
	π_3	0.0	0.2	**99.8**

No Noise

Table B-23. Confusion Matrices for Simulated Modulated Signals (Three-Class, Twenty-Six Features): Perceptron. (see App B cover page for table description)

INDEX: 55.0	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**39.5**	7.8	0.8
π_2	10.3	**30.2**	0.4
π_3	4.8	0	**95.2**

SNR = 20 dB

INDEX: 58.2	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**47.1**	9.9	8.0
π_2	25.6	**29.7**	8.3
π_3	1.6	0.6	**97.8**

SNR = 15 dB

INDEX: 56.1	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**24.6**	29.7	38.6
π_2	13.6	**46.1**	35.0
π_3	1.1	1.8	**97.1**

SNR = 10 dB

INDEX: 52.3	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**24.1**	38.2	22.4
π_2	18.5	**46.1**	21.4
π_3	3.1	9.7	**86.7**

SNR = 5 dB

INDEX: 41.2	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**30.6**	23.3	37.5
π_2	30.2	**23.8**	37.3
π_3	15.5	12.5	**69.3**

SNR = 0 dB

INDEX: 29.6	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**10.6**	30.5	42.2
π_2	10.5	**30.8**	42.5
π_3	8.6	32.3	**47.5**

SNR = -5 dB

INDEX: 30.4	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**10.5**	28.2	51.3
π_2	10.5	**28.7**	51.0
π_3	10.4	28.7	**51.8**

SNR = -10 dB

INDEX: 27.8	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**21.4**	23.9	37.2
π_2	21.3	**23.9**	37.2
π_3	21.3	23.5	**37.9**

SNR = -15 dB

INDEX: 29.9	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**11.4**	37.1	40.9
π_2	11.0	**37.4**	41.0
π_3	11.2	37.2	**40.9**

SNR = -20 dB

INDEX: 83.8	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**74.1**	4.1	19.5
π_2	7.3	**78.1**	6.2
π_3	0.7	0.0	**99.3**

No Noise

Table B-24. Confusion Matrices for Simulated Modulated Signals (Three-Class, Eleven-Features): Perceptron. (see App B cover page for table description)

INDEX: 89.0	SELECTED		
	π_1*	π_2*	π_3*
ACTUAL π_1	**86.1**	13.9	0.0
ACTUAL π_2	19.1	**80.9**	0.0
ACTUAL π_3	0.0	0.0	**100**

SNR = 20 dB

INDEX: 82.6	SELECTED		
	π_1*	π_2*	π_3*
ACTUAL π_1	**75.0**	25.0	0.0
ACTUAL π_2	27.0	**73.0**	0.0
ACTUAL π_3	0.1	0.1	**99.8**

SNR = 15 dB

INDEX: 76.1	SELECTED		
	π_1*	π_2*	π_3*
ACTUAL π_1	**67.7**	32.3	0.1
ACTUAL π_2	38.7	**61.2**	0.1
ACTUAL π_3	0.2	0.4	**99.5**

SNR = 10 dB

INDEX: 70.4	SELECTED		
	π_1*	π_2*	π_3*
ACTUAL π_1	**55.4**	42.9	1.7
ACTUAL π_2	38.5	**58.8**	2.6
ACTUAL π_3	1.1	1.9	**96.9**

SNR = 5 dB

INDEX: 56.3	SELECTED		
	π_1*	π_2*	π_3*
ACTUAL π_1	**44.0**	43.7	12.3
ACTUAL π_2	42.1	**44.9**	13.0
ACTUAL π_3	10.6	9.4	**80.0**

SNR = 0 dB

INDEX: 42.5	SELECTED		
	π_1*	π_2*	π_3*
ACTUAL π_1	**35.7**	37.0	27.2
ACTUAL π_2	34.5	**36.7**	28.9
ACTUAL π_3	21.3	23.5	**55.2**

SNR = -5 dB

INDEX: 34.7	SELECTED		
	π_1*	π_2*	π_3*
ACTUAL π_1	**33.4**	34.4	32.2
ACTUAL π_2	34.1	**34.8**	31.1
ACTUAL π_3	31.5	32.7	**35.8**

SNR = -10 dB

INDEX: 33.5	SELECTED		
	π_1*	π_2*	π_3*
ACTUAL π_1	**35.0**	34.6	30.4
ACTUAL π_2	34.4	**35.0**	30.6
ACTUAL π_3	35.2	34.4	**30.4**

SNR = -15 dB

INDEX: 33.7	SELECTED		
	π_1*	π_2*	π_3*
ACTUAL π_1	**31.7**	33.7	34.6
ACTUAL π_2	32.4	**34.1**	33.4
ACTUAL π_3	31.8	32.7	**35.5**

SNR = -20 dB

INDEX: 94.8	SELECTED		
	π_1*	π_2*	π_3*
ACTUAL π_1	**93.7**	6.3	0.0
ACTUAL π_2	9.2	**90.8**	0.0
ACTUAL π_3	0.0	0.1	**99.9**

No Noise

Table B-25. Confusion Matrices for Simulated Modulated Signals (Three-Class, Fifty-One Features): MSNN. (see App B cover page for table description)

INDEX: 86.0	SELECTED π_1^*	π_2^*	π_3^*
ACTUAL π_1	**76.9**	23.1	0.0
π_2	18.9	**81.1**	0.0
π_3	0.0	0.0	**100**

SNR = 20 dB

INDEX: 81.5	SELECTED π_1^*	π_2^*	π_3^*
ACTUAL π_1	**72.6**	27.4	0.0
π_2	27.9	**72.1**	0.0
π_3	0.0	0.1	**99.9**

SNR = 15 dB

INDEX: 77.7	SELECTED π_1^*	π_2^*	π_3^*
ACTUAL π_1	**70.4**	29.4	0.2
π_2	36.7	**63.0**	0.3
π_3	0.2	0.1	**99.7**

SNR = 10 dB

INDEX: 69.8	SELECTED π_1^*	π_2^*	π_3^*
ACTUAL π_1	**57.7**	41.4	1.0
π_2	41.8	**56.8**	1.5
π_3	2.4	2.8	**94.8**

SNR = 5 dB

INDEX: 56.2	SELECTED π_1^*	π_2^*	π_3^*
ACTUAL π_1	**44.5**	42.2	13.4
π_2	41.6	**44.9**	13.5
π_3	10.4	10.5	**79.2**

SNR = 0 dB

INDEX: 41.7	SELECTED π_1^*	π_2^*	π_3^*
ACTUAL π_1	**36.5**	37.2	26.3
π_2	35.5	**36.6**	28.0
π_3	22.6	25.5	**51.9**

SNR = -5 dB

INDEX: 34.8	SELECTED π_1^*	π_2^*	π_3^*
ACTUAL π_1	**34.4**	32.4	33.2
π_2	34.4	**33.1**	32.5
π_3	32.0	31.1	**37.0**

SNR = -10 dB

INDEX: 33.4	SELECTED π_1^*	π_2^*	π_3^*
ACTUAL π_1	**31.6**	33.8	34.6
π_2	32.3	**32.8**	34.9
π_3	32.7	31.4	**35.8**

SNR = -15 dB

INDEX: 33.1	SELECTED π_1^*	π_2^*	π_3^*
ACTUAL π_1	**36.8**	32.9	30.3
π_2	37.1	**32.8**	30.1
π_3	36.5	33.9	**29.7**

SNR = -20 dB

INDEX: 93.8	SELECTED π_1^*	π_2^*	π_3^*
ACTUAL π_1	**92.5**	7.5	0.0
π_2	11.1	**88.9**	0.0
π_3	0.0	0.0	**100**

No Noise

Table B-26. Confusion Matrices for Simulated Modulated Signals (Three-Class, Twenty-Six Features): MSNN. (see App B cover page for table description)

INDEX: 81.8	SELECTED π_1^*	π_2^*	π_3^*
ACTUAL π_1	**75.7**	24.2	0.1
π_2	30.3	**69.6**	0.0
π_3	0.0	0.0	**100**

SNR = 20 dB

INDEX: 78.7	SELECTED π_1^*	π_2^*	π_3^*
ACTUAL π_1	**68.5**	31.4	0.1
π_2	32.2	**67.7**	0.0
π_3	0.2	0.0	**99.8**

SNR = 15 dB

INDEX: 75.8	SELECTED π_1^*	π_2^*	π_3^*
ACTUAL π_1	**63.6**	35.9	0.5
π_2	34.4	**65.2**	0.4
π_3	0.5	0.7	**98.7**

SNR = 10 dB

INDEX: 68.0	SELECTED π_1^*	π_2^*	π_3^*
ACTUAL π_1	**55.2**	40.4	4.4
π_2	39.6	**56.2**	4.2
π_3	3.1	4.3	**92.6**

SNR = 5 dB

INDEX: 55.3	SELECTED π_1^*	π_2^*	π_3^*
ACTUAL π_1	**46.9**	37.9	15.2
π_2	42.9	**41.4**	15.8
π_3	10.5	12.0	**77.5**

SNR = 0 dB

INDEX: 41.1	SELECTED π_1^*	π_2^*	π_3^*
ACTUAL π_1	**34.7**	32.8	32.5
π_2	34.6	**35.1**	30.3
π_3	23.4	23.1	**53.5**

SNR = -5 dB

INDEX: 34.5	SELECTED π_1^*	π_2^*	π_3^*
ACTUAL π_1	**30.1**	35.8	34.1
π_2	29.0	**36.1**	34.9
π_3	29.0	33.6	**37.4**

SNR = -10 dB

INDEX: 34.2	SELECTED π_1^*	π_2^*	π_3^*
ACTUAL π_1	**30.7**	36.2	33.2
π_2	28.7	**39.2**	32.1
π_3	30.7	36.4	**32.9**

SNR = -15 dB

INDEX: 33.2	SELECTED π_1^*	π_2^*	π_3^*
ACTUAL π_1	**36.6**	29.0	34.4
π_2	37.7	**29.5**	32.8
π_3	36.4	30.3	**33.3**

SNR = -20 dB

INDEX: 94.3	SELECTED π_1^*	π_2^*	π_3^*
ACTUAL π_1	**93.6**	6.3	0.0
π_2	10.6	**89.4**	0.0
π_3	0.0	0.0	**100**

No Noise

Table B-27. Confusion Matrices for Simulated Modulated Signals (Three-Class, Eleven-Features): MSNN. (see App B cover page for table description)

INDEX: 72.9		SELECTED π_1*	π_2*	π_3*
ACTUAL	π_1	**58.7**	40.8	0.5
	π_2	36.5	**63.3**	0.2
	π_3	2.0	1.2	**96.8**

SNR = 20 dB

INDEX: 68.7		SELECTED π_1*	π_2*	π_3*
ACTUAL	π_1	**53.9**	46.0	0.1
	π_2	42.4	**57.6**	0.0
	π_3	4.2	1.2	**94.5**

SNR = 15 dB

INDEX: 64.1		SELECTED π_1*	π_2*	π_3*
ACTUAL	π_1	**47.3**	52.6	0.0
	π_2	38.6	**61.3**	0.1
	π_3	4.7	11.5	**83.8**

SNR = 10 dB

INDEX: 55.3		SELECTED π_1*	π_2*	π_3*
ACTUAL	π_1	**47.6**	49.0	3.4
	π_2	45.7	**50.2**	4.1
	π_3	14.5	17.4	**68.2**

SNR = 5 dB

INDEX: 43.9		SELECTED π_1*	π_2*	π_3*
ACTUAL	π_1	**36.4**	42.8	20.8
	π_2	36.2	**42.8**	21.0
	π_3	22.5	25.0	**52.5**

SNR = 0 dB

INDEX: 35.0		SELECTED π_1*	π_2*	π_3*
ACTUAL	π_1	**32.4**	37.1	30.5
	π_2	32.8	**37.2**	30.0
	π_3	29.4	35.3	**35.3**

SNR = -5 dB

INDEX: 33.4		SELECTED π_1*	π_2*	π_3*
ACTUAL	π_1	**34.4**	31.5	34.1
	π_2	34.9	**31.4**	33.7
	π_3	35.7	30.0	**34.3**

SNR = -10 dB

INDEX: 33.4		SELECTED π_1*	π_2*	π_3*
ACTUAL	π_1	**33.6**	29.7	36.6
	π_2	33.4	**29.9**	36.7
	π_3	33.6	29.7	**36.7**

SNR = -15 dB

INDEX: 33.2		SELECTED π_1*	π_2*	π_3*
ACTUAL	π_1	**33.4**	32.2	34.3
	π_2	34.4	**32.2**	33.4
	π_3	33.6	32.3	**34.1**

SNR = -20 dB

INDEX: 63.0		SELECTED π_1*	π_2*	π_3*
ACTUAL	π_1	**3.7**	82.6	13.7
	π_2	0.9	**85.4**	13.7
	π_3	0.0	0.2	**99.8**

No Noise

Table B-28. Confusion Matrices for Simulated Modulated Signals (Three-Class, Fifty-One Features): MSNN Mod 1. (see App B cover page for table description)

INDEX: 71.4	SELECTED π_1^*	π_2^*	π_3^*
ACTUAL π_1	**48.1**	51.5	0.3
π_2	31.3	**68.2**	0.4
π_3	1.3	0.7	**97.9**

SNR = 20 dB

INDEX: 68.9	SELECTED π_1^*	π_2^*	π_3^*
ACTUAL π_1	**50.5**	49.4	0.1
π_2	38.1	**61.8**	0.1
π_3	3.4	2.1	**94.5**

SNR = 15 dB

INDEX: 66.3	SELECTED π_1^*	π_2^*	π_3^*
ACTUAL π_1	**46.8**	52.7	0.4
π_2	34.2	**65.3**	0.5
π_3	5.2	8.1	**86.7**

SNR = 10 dB

INDEX: 57.7	SELECTED π_1^*	π_2^*	π_3^*
ACTUAL π_1	**49.1**	48.6	2.4
π_2	47.0	**50.2**	2.8
π_3	11.9	14.3	**73.9**

SNR = 5 dB

INDEX: 43.6	SELECTED π_1^*	π_2^*	π_3^*
ACTUAL π_1	**39.4**	42.6	17.9
π_2	39.5	**42.8**	17.7
π_3	26.2	25.1	**48.7**

SNR = 0 dB

INDEX: 35.5	SELECTED π_1^*	π_2^*	π_3^*
ACTUAL π_1	**33.4**	31.5	35.1
π_2	32.6	**32.5**	34.9
π_3	30.2	29.3	**40.5**

SNR = -5 dB

INDEX: 34.0	SELECTED π_1^*	π_2^*	π_3^*
ACTUAL π_1	**32.3**	33.0	34.7
π_2	31.7	**33.7**	34.6
π_3	31.2	32.6	**36.1**

SNR = -10 dB

INDEX: 33.4	SELECTED π_1^*	π_2^*	π_3^*
ACTUAL π_1	**33.5**	33.5	33.0
π_2	33.9	**33.4**	32.7
π_3	33.2	33.7	**33.1**

SNR = -15 dB

INDEX: 33.3	SELECTED π_1^*	π_2^*	π_3^*
ACTUAL π_1	**32.9**	32.4	34.6
π_2	33.3	**32.3**	34.4
π_3	32.6	32.6	**34.8**

SNR = -20 dB

INDEX: 64.4	SELECTED π_1^*	π_2^*	π_3^*
ACTUAL π_1	**0.0**	90.6	9.4
π_2	0.0	**93.1**	6.9
π_3	0.0	0.0	**100**

No Noise

Table B-29. Confusion Matrices for Simulated Modulated Signals (Three-Class, Twenty-Six Features): MSNN Mod 1. (see App B cover page for table description)

INDEX: 72.2	SELECTED π_1^*	π_2^*	π_3^*
ACTUAL π_1	**72.2**	26.6	1.2
π_2	53.1	**46.4**	0.6
π_3	1.6	0.6	**97.8**

SNR = 20 dB

INDEX: 71.2	SELECTED π_1^*	π_2^*	π_3^*
ACTUAL π_1	**59.1**	40.3	0.6
π_2	41.9	**57.8**	0.4
π_3	1.6	1.6	**96.9**

SNR = 15 dB

INDEX: 65.1	SELECTED π_1^*	π_2^*	π_3^*
ACTUAL π_1	**55.6**	44.3	0.1
π_2	44.4	**55.6**	0.0
π_3	8.1	7.9	**84.1**

SNR = 10 dB

INDEX: 57.9	SELECTED π_1^*	π_2^*	π_3^*
ACTUAL π_1	**48.1**	50.8	1.1
π_2	41.1	**57.0**	1.9
π_3	11.2	20.2	**68.7**

SNR = 5 dB

INDEX: 47.7	SELECTED π_1^*	π_2^*	π_3^*
ACTUAL π_1	**41.4**	41.0	17.6
π_2	39.5	**42.0**	18.6
π_3	20.0	20.1	**59.9**

SNR = 0 dB

INDEX: 35.7	SELECTED π_1^*	π_2^*	π_3^*
ACTUAL π_1	**37.6**	32.4	30.1
π_2	37.7	**32.6**	29.7
π_3	30.8	32.2	**37.0**

SNR = -5 dB

INDEX: 33.6	SELECTED π_1^*	π_2^*	π_3^*
ACTUAL π_1	**34.2**	32.9	32.9
π_2	33.5	**33.4**	33.1
π_3	33.7	33.2	**33.2**

SNR = -10 dB

INDEX: 33.5	SELECTED π_1^*	π_2^*	π_3^*
ACTUAL π_1	**32.8**	31.3	35.8
π_2	32.6	**31.4**	36.1
π_3	33.1	30.6	**36.3**

SNR = -15 dB

INDEX: 33.3	SELECTED π_1^*	π_2^*	π_3^*
ACTUAL π_1	**29.7**	35.6	34.7
π_2	29.7	**35.6**	34.7
π_3	30.4	35.1	**34.5**

SNR = -20 dB

INDEX: 45.1	SELECTED π_1^*	π_2^*	π_3^*
ACTUAL π_1	**1.9**	16.5	81.6
π_2	0.7	**33.3**	66.0
π_3	0.0	0.0	**100**

No Noise

Table B-30. Confusion Matrices for Simulated Modulated Signals (Three-Class, Eleven-Features): MSNN Mod 1. (see App B cover page for table description)

INDEX: 87.3	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**79.5**	20.5	0.0
π_2	16.6	**83.0**	0.3
π_3	0.3	0.3	**99.4**

SNR = 20 dB

INDEX: 81.3	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**70.6**	29.3	0.1
π_2	24.8	**74.9**	0.4
π_3	0.9	0.7	**98.4**

SNR = 15 dB

INDEX: 74.0	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**61.8**	37.9	0.3
π_2	37.8	**62.0**	0.2
π_3	0.9	0.8	**98.4**

SNR = 10 dB

INDEX: 67.4	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**52.8**	43.7	3.5
π_2	41.0	**53.7**	5.3
π_3	1.4	2.8	**95.8**

SNR = 5 dB

INDEX: 55.8	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**38.7**	46.8	14.4
π_2	37.9	**47.0**	15.1
π_3	9.8	8.5	**81.8**

SNR = 0 dB

INDEX: 41.9	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**37.1**	34.7	28.2
π_2	35.7	**34.4**	29.9
π_3	21.8	24.0	**54.2**

SNR = -5 dB

INDEX: 34.0	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**31.2**	34.5	34.3
π_2	32.3	**34.2**	33.5
π_3	30.0	33.4	**36.6**

SNR = -10 dB

INDEX: 33.1	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**46.8**	27.5	25.7
π_2	46.8	**27.5**	25.7
π_3	47.2	27.8	**25.0**

SNR = -15 dB

INDEX: 33.5	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**34.3**	31.7	34.1
π_2	34.9	**31.6**	33.5
π_3	34.0	31.3	**34.7**

SNR = -20 dB

INDEX: 92.8	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**88.8**	11.2	0.0
π_2	9.7	**90.3**	0.0
π_3	0.4	0.3	**99.3**

No Noise

Table B-31. Confusion Matrices for Simulated Modulated Signals (Three-Class, Fifty-One Features): MSNN Mod 2. (see App B cover page for table description)

INDEX: 83.2	SELECTED π_1*	SELECTED π_2*	SELECTED π_3*
ACTUAL π_1	**75.7**	24.3	0.0
ACTUAL π_2	25.8	**74.1**	0.1
ACTUAL π_3	0.0	0.1	**99.9**

SNR = 20 dB

INDEX: 80.8	SELECTED π_1*	SELECTED π_2*	SELECTED π_3*
ACTUAL π_1	**71.3**	28.6	0.0
ACTUAL π_2	28.7	**71.1**	0.2
ACTUAL π_3	0.0	0.1	**99.9**

SNR = 15 dB

INDEX: 77.7	SELECTED π_1*	SELECTED π_2*	SELECTED π_3*
ACTUAL π_1	**68.4**	31.6	0.0
ACTUAL π_2	34.7	**65.2**	0.1
ACTUAL π_3	0.2	0.3	**99.5**

SNR = 10 dB

INDEX: 66.0	SELECTED π_1*	SELECTED π_2*	SELECTED π_3*
ACTUAL π_1	**58.4**	38.5	3.1
ACTUAL π_2	43.7	**51.5**	4.9
ACTUAL π_3	6.6	5.2	**88.2**

SNR = 5 dB

INDEX: 55.0	SELECTED π_1*	SELECTED π_2*	SELECTED π_3*
ACTUAL π_1	**48.0**	39.8	12.2
ACTUAL π_2	46.3	**41.7**	12.0
ACTUAL π_3	12.9	11.9	**75.3**

SNR = 0 dB

INDEX: 42.3	SELECTED π_1*	SELECTED π_2*	SELECTED π_3*
ACTUAL π_1	**28.3**	41.0	30.7
ACTUAL π_2	26.5	**40.5**	33.0
ACTUAL π_3	16.6	25.2	**58.3**

SNR = -5 dB

INDEX: 34.1	SELECTED π_1*	SELECTED π_2*	SELECTED π_3*
ACTUAL π_1	**26.6**	38.9	34.5
ACTUAL π_2	27.2	**38.5**	34.3
ACTUAL π_3	25.7	37.2	**37.1**

SNR = -10 dB

INDEX: 33.5	SELECTED π_1*	SELECTED π_2*	SELECTED π_3*
ACTUAL π_1	**32.8**	28.4	38.8
ACTUAL π_2	33.1	**27.8**	39.1
ACTUAL π_3	33.8	26.3	**39.9**

SNR = -15 dB

INDEX: 33.2	SELECTED π_1*	SELECTED π_2*	SELECTED π_3*
ACTUAL π_1	**39.5**	25.4	35.1
ACTUAL π_2	40.4	**24.7**	34.9
ACTUAL π_3	39.5	25.2	**35.3**

SNR = -20 dB

INDEX: 92.2	SELECTED π_1*	SELECTED π_2*	SELECTED π_3*
ACTUAL π_1	**84.6**	14.9	0.5
ACTUAL π_2	7.8	**92.1**	0.1
ACTUAL π_3	0.0	0.1	**99.8**

No Noise

Table B-32. Confusion Matrices for Simulated Modulated Signals (Three-Class, Twenty-Six Features): MSNN Mod 2. (see App B cover page for table description)

INDEX: 81.1	SELECTED π_1^*	π_2^*	π_3^*
ACTUAL π_1	**68.8**	28.3	3.0
π_2	24.9	**74.8**	0.3
π_3	0.3	0.0	**99.7**

SNR = 20 dB

INDEX: 77.3	SELECTED π_1^*	π_2^*	π_3^*
ACTUAL π_1	**61.6**	38.1	0.3
π_2	28.9	**71.0**	0.0
π_3	0.6	0.2	**99.2**

SNR = 15 dB

INDEX: 75.1	SELECTED π_1^*	π_2^*	π_3^*
ACTUAL π_1	**67.9**	31.3	0.8
π_2	40.1	**59.5**	0.4
π_3	1.0	1.2	**97.8**

SNR = 10 dB

INDEX: 64.4	SELECTED π_1^*	π_2^*	π_3^*
ACTUAL π_1	**53.9**	39.8	6.3
π_2	40.8	**53.5**	5.6
π_3	6.0	8.1	**85.9**

SNR = 5 dB

INDEX: 55.2	SELECTED π_1^*	π_2^*	π_3^*
ACTUAL π_1	**47.1**	36.8	16.1
π_2	43.2	**40.1**	16.7
π_3	9.6	12.1	**78.3**

SNR = 0 dB

INDEX: 40.8	SELECTED π_1^*	π_2^*	π_3^*
ACTUAL π_1	**28.4**	39.2	32.4
π_2	29.4	**40.3**	30.3
π_3	20.6	25.6	**53.7**

SNR = -5 dB

INDEX: 34.2	SELECTED π_1^*	π_2^*	π_3^*
ACTUAL π_1	**27.1**	34.6	38.3
π_2	26.9	**33.9**	39.2
π_3	25.5	32.8	**41.6**

SNR = -10 dB

INDEX: 33.6	SELECTED π_1^*	π_2^*	π_3^*
ACTUAL π_1	**24.5**	43.3	32.2
π_2	23.7	**44.2**	32.1
π_3	24.2	43.7	**32.2**

SNR = -15 dB

INDEX: 33.4	SELECTED π_1^*	π_2^*	π_3^*
ACTUAL π_1	**39.3**	25.9	34.8
π_2	39.7	**25.9**	34.5
π_3	38.6	26.4	**35.0**

SNR = -20 dB

INDEX: 91.4	SELECTED π_1^*	π_2^*	π_3^*
ACTUAL π_1	**84.7**	13.1	2.2
π_2	10.4	**89.5**	0.0
π_3	0.1	0.0	**99.9**

No Noise

Table B-33. Confusion Matrices for Simulated Modulated Signals (Three-Class, Eleven-Features): MSNN Mod 2. (see App B cover page for table description)

INDEX: 88.7		SELECTED		
		π_1*	π_2*	π_3*
ACTUAL	π_1	**85.9**	14.1	0.0
	π_2	19.5	**80.5**	0.0
	π_3	0.0	0.3	**99.7**

SNR = 20 dB

INDEX: 82.5		SELECTED		
		π_1*	π_2*	π_3*
ACTUAL	π_1	**74.3**	25.7	0.0
	π_2	26.6	**73.4**	0.0
	π_3	0.1	0.1	**99.8**

SNR = 15 dB

INDEX: 75.4		SELECTED		
		π_1*	π_2*	π_3*
ACTUAL	π_1	**66.3**	32.0	1.7
	π_2	38.2	**60.4**	1.4
	π_3	0.1	0.4	**99.5**

SNR = 10 dB

INDEX: 70.2		SELECTED		
		π_1*	π_2*	π_3*
ACTUAL	π_1	**54.7**	43.5	1.7
	π_2	38.7	**58.6**	2.7
	π_3	1.1	1.8	**97.1**

SNR = 5 dB

INDEX: 56.2		SELECTED		
		π_1*	π_2*	π_3*
ACTUAL	π_1	**44.5**	42.9	12.6
	π_2	42.7	**44.1**	13.2
	π_3	10.8	9.1	**80.1**

SNR = 0 dB

INDEX: 42.3		SELECTED		
		π_1*	π_2*	π_3*
ACTUAL	π_1	**37.1**	36.3	26.6
	π_2	35.9	**35.8**	28.4
	π_3	22.1	23.8	**54.1**

SNR = -5 dB

INDEX: 34.8		SELECTED		
		π_1*	π_2*	π_3*
ACTUAL	π_1	**33.3**	35.3	31.3
	π_2	33.9	**35.7**	30.4
	π_3	31.3	33.3	**35.4**

SNR = -10 dB

INDEX: 33.6		SELECTED		
		π_1*	π_2*	π_3*
ACTUAL	π_1	**34.9**	33.9	31.2
	π_2	34.4	**34.5**	31.1
	π_3	34.9	33.8	**31.3**

SNR = -15 dB

INDEX: 33.7		SELECTED		
		π_1*	π_2*	π_3*
ACTUAL	π_1	**31.8**	34.7	33.5
	π_2	32.6	**35.2**	32.2
	π_3	31.8	34.0	**34.2**

SNR = -20 dB

INDEX: 94.0		SELECTED		
		π_1*	π_2*	π_3*
ACTUAL	π_1	**94.8**	5.2	0.0
	π_2	12.8	**87.2**	0.0
	π_3	0.0	0.1	**99.9**

No Noise

Table B-34. Confusion Matrices for Simulated Modulated Signals (Three-Class, Fifty-One Features): MSNN Mod 3. (see App B cover page for table description)

INDEX: 85.8	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**77.6**	22.3	0.0
ACTUAL π_2	20.0	**79.9**	0.0
ACTUAL π_3	0.0	0.0	**99.9**

SNR = 20 dB

INDEX: 81.2	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**73.0**	27.0	0.0
ACTUAL π_2	28.8	**71.2**	0.0
ACTUAL π_3	0.0	0.5	**99.5**

SNR = 15 dB

INDEX: 77.6	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**69.5**	30.4	0.2
ACTUAL π_2	36.1	**63.7**	0.2
ACTUAL π_3	0.2	0.3	**99.5**

SNR = 10 dB

INDEX: 69.8	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**56.1**	42.9	0.9
ACTUAL π_2	40.6	**58.0**	1.4
ACTUAL π_3	2.1	2.5	**95.4**

SNR = 5 dB

INDEX: 56.0	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**45.5**	42.2	12.3
ACTUAL π_2	42.9	**44.4**	12.7
ACTUAL π_3	11.0	10.9	**78.1**

SNR = 0 dB

INDEX: 41.7	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**37.2**	36.6	26.2
ACTUAL π_2	35.9	**36.2**	27.9
ACTUAL π_3	22.6	25.8	**51.7**

SNR = -5 dB

INDEX: 34.8	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**33.8**	33.1	33.1
ACTUAL π_2	34.1	**33.5**	32.5
ACTUAL π_3	31.4	31.4	**37.2**

SNR = -10 dB

INDEX: 33.5	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**31.4**	34.1	34.5
ACTUAL π_2	31.9	**33.4**	34.7
ACTUAL π_3	32.6	31.7	**35.7**

SNR = -15 dB

INDEX: 33.0	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**36.6**	32.8	30.6
ACTUAL π_2	37.2	**32.4**	30.4
ACTUAL π_3	36.5	33.6	**29.9**

SNR = -20 dB

INDEX: 93.1	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**92.8**	7.2	0.0
ACTUAL π_2	13.3	**86.7**	0.0
ACTUAL π_3	0.0	0.1	**99.9**

No Noise

Table B-35. Confusion Matrices for Simulated Modulated Signals (Three-Class, Twenty-Six Features): MSNN Mod 3. (see App B cover page for table description)

INDEX: 81.0	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**73.5**	26.3	0.2
π_2	28.3	**71.7**	0.0
π_3	2.1	0.0	**97.9**

SNR = 20 dB

INDEX: 78.5	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**68.4**	31.5	0.1
π_2	32.4	**67.6**	0.0
π_3	0.2	0.2	**99.6**

SNR = 15 dB

INDEX: 75.9	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**63.9**	35.8	0.4
π_2	34.7	**65.0**	0.3
π_3	0.5	0.7	**98.8**

SNR = 10 dB

INDEX: 68.0	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**55.3**	40.8	3.9
π_2	40.1	**56.1**	3.8
π_3	3.1	4.4	**92.6**

SNR = 5 dB

INDEX: 55.6	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**46.6**	38.3	15.1
π_2	41.7	**42.4**	15.9
π_3	10.5	11.7	**77.8**

SNR = 0 dB

INDEX: 40.9	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**35.1**	32.4	32.5
π_2	35.6	**34.3**	30.0
π_3	24.0	22.8	**53.3**

SNR = -5 dB

INDEX: 34.5	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**33.5**	32.7	33.8
π_2	32.4	**32.5**	35.1
π_3	31.3	31.2	**37.6**

SNR = -10 dB

INDEX: 34.3	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**30.9**	36.1	33.0
π_2	29.0	**39.0**	32.0
π_3	30.7	36.2	**33.1**

SNR = -15 dB

INDEX: 32.8	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**35.8**	30.9	33.3
π_2	37.5	**30.5**	32.0
π_3	35.9	31.9	**32.2**

SNR = -20 dB

INDEX: 92.6	SELECTED π_1*	π_2*	π_3*
ACTUAL π_1	**92.8**	7.2	0.0
π_2	13.4	**86.6**	0.0
π_3	1.6	0.0	**98.4**

No Noise

Table B-36. Confusion Matrices for Simulated Modulated Signals (Three-Class, Eleven-Features): MSNN Mod 3. (see App B cover page for table description)

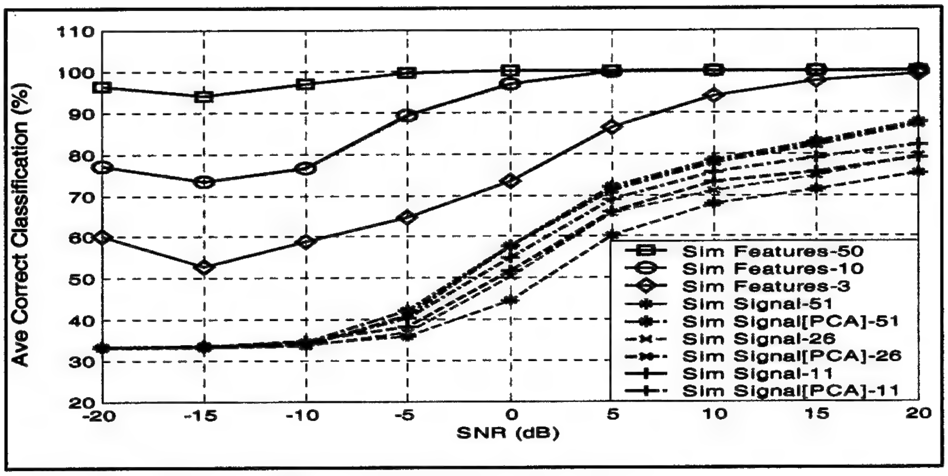


Figure B-1. Statistical Classifier Performance Results.

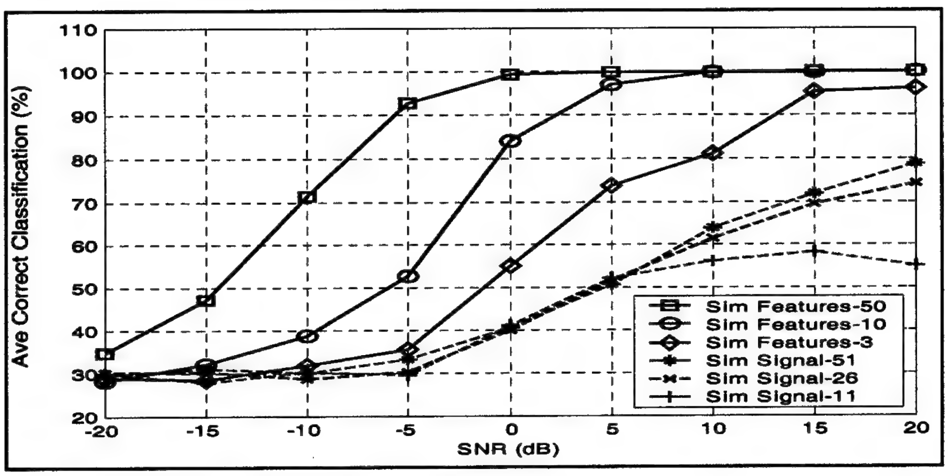


Figure B-2. Perceptron Performance Results.

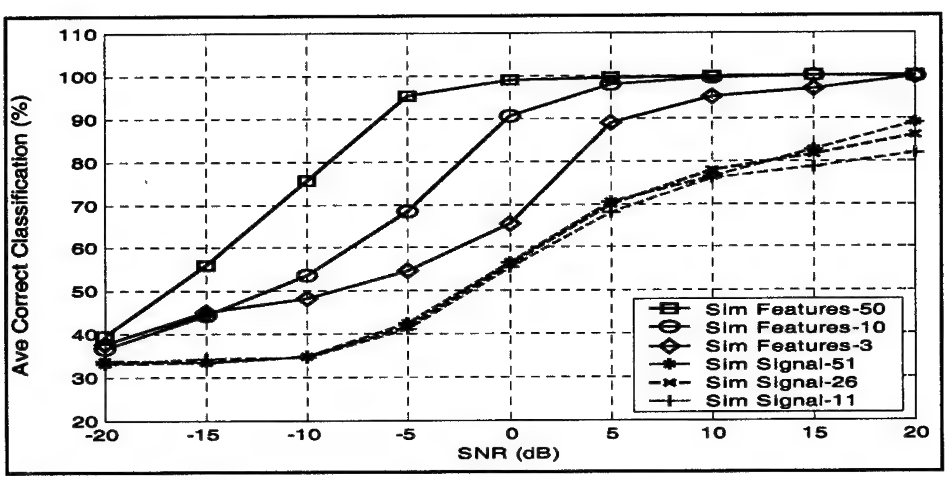


Figure B-3. MSNN Performance Results.

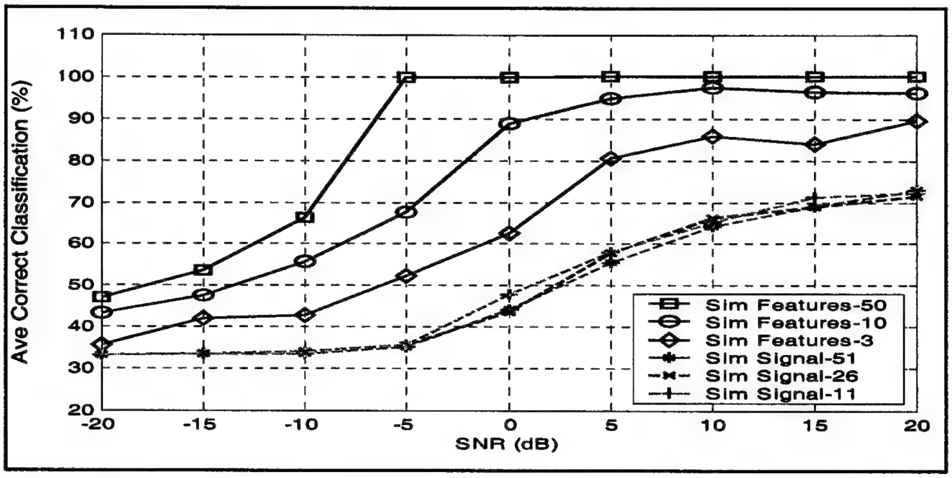


Figure B-4. MSNN Mod 1 Performance Results.

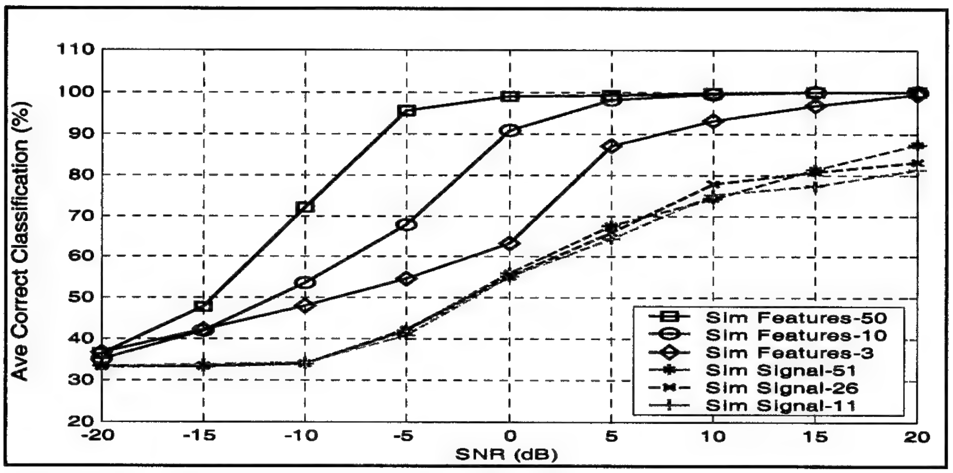


Figure B-5. MSNN Mod 2 Performance Results.

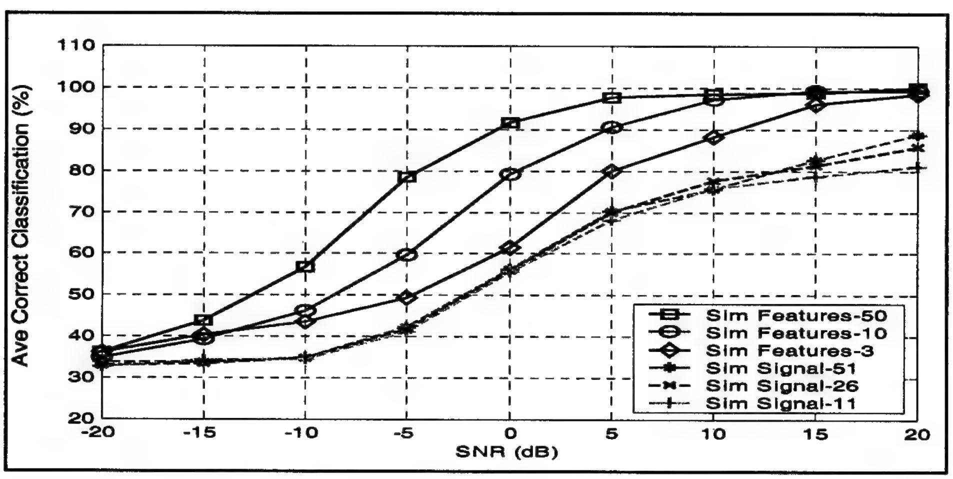


Figure B-6. MSNN Mod 3 Performance Results.

APPENDIX C. MATLAB CLASSIFICATION PROGRAMS

This section contains the MATLAB programs used to generate the simulation results discussed in Chapters IV and V. These functions are categorized as either common or specific to a particular classification scheme.

A. COMMON PROGRAMS

The common programs included in this section include the main program; feature simulation functions; modulated signal simulation and feature extraction functions; and data conditioning and display routines.

1. Controlling Program: *simmsnn_compare.m*

```
%*****************************************************************************************
% COMPARE classification methods
%
% 5 March 2000
% Miguel G. San Pedro
%*****************************************************************************************
clear
format compact
format short e

global gloUsrReq
gloUsrReq = input('Skip all optional displays (Y/N): ','s');

global gloUsrPlot
gloUsrPlot = input('Plot learning curves (Y/N):            ','s');

num_data = [];               % number of training realizations
class_mean = [];             % feature mean values
class_cov = [];              % feature covariance matrix
class_var = [];              % feature variance values
classData = [];              % training data set
testClass = [];              % testing data set
snr = [];                    % training/testing signal SNR

save test\testClass.dat testClass -ascii -tabs

% ASK if simulate signal or simulate data
usrReq = input('Simulate <signal> or <*data*>:              ','s');
disp(' ')

% GENERATE testing/training data
if (usrReq == 'signal')
   num_class = 3;            % number of signal classes
   A = 4;                    % SET signal amplitude
   T = 1e-7;                 % SET bit period (sec)
   fs = 5e8;                 % SET bit sampling frequency (samples/sec)
   fc = 4e7;                 % SET carrier frequency (Hz_
   n = linspace(0,T,fs*T);
   features = [];            % vector of distinguishing features
   trnFeatures = [];         % vector of class distinguishing features
   mnFeatures = [];          % vector of class distinguishing feature mean
   covFeatures = [];         % class covariance matrix
```

```
    varFeatures = [];          % vector of class distinguishing feature variance

    % DETERMINE signal features
    disp('EXTRACTING SIGNAL FEATURES.......')
    features = detFeatures;
    [numRows,num_features] = size(features);
    disp(['Number of features: ',num2str(num_features)])
    disp (' ')

    % GENERATE signals
    num_data = input('Enter number of training signals (def=100):    ');
    if (isempty(num_data))
       num_data = 100;
    end

    usrSNR = input('Add noise (Y/N):   ','s');
    if (usrSNR == 'Y')
       snr = input('Enter signal SNR (default=0dB):   ');
       disp(' ')
       if (isempty(snr));
          snr = 0;
       end
    else
       snr = 9999;
    end

    plotSignal('plot2ASK',A,T,fc,n,features,snr)
    plotSignal('plot2PSK',A,T,fc,n,features,snr)
    plotSignal('plot2FSK',A,T,fc,n,features,snr)

    [trnFeatures,mnFeatures,covFeatures,varFeatures]...
         = genSignal('gen2ASK',num_data,A,T,fs,n,features,snr);
    classData = [classData;trnFeatures];
    class_mean = [class_mean mnFeatures];
    class_cov = [class_cov covFeatures];
    class_var = [class_var varFeatures];

    [trnFeatures,mnFeatures,covFeatures,varFeatures]...
         = genSignal('gen2PSK',num_data,A,T,fs,n,features,snr);
    classData = [classData;trnFeatures];
    class_mean = [class_mean mnFeatures];
    class_cov = [class_cov covFeatures];
    class_var = [class_var varFeatures];

    [trnFeatures,mnFeatures,covFeatures,varFeatures]...
         = genSignal('gen2FSK',num_data,A,T,fs,n,features,snr);
    classData = [classData;trnFeatures];
    class_mean = [class_mean mnFeatures];
    class_cov = [class_cov covFeatures];
    class_var = [class_var varFeatures];

    % GENERATE random test data
    load testClass.dat
    randTest = 100*randn(num_features,num_data*10);
    testClass = [testClass;randTest];
    save test\testClass.dat testClass -ascii -tabs

else
    % ASK user for input data; else set default values
    num_class = [];            % number of signal classes
    num_features = [];         % number of distinguishing features

    userInput = input('Enter user defined inputs (Y/N):      ','s');
    if (userInput == 'Y')
       disp(' ')
       [num_data,num_class,num_features,class_mean,class_var]...
          = userData(num_data,num_class,num_features,class_mean,class_var);
```

```
   else
      % default values
      num_data = 100;
      num_class = 3;
      num_features = 3;
      class_mean = 2*rand(num_features,num_class) - 1;

      usrSNR = input('Add noise (Y/N):          ','s');
      if (usrSNR == 'Y')
         snr = input('Enter feature SNR (default=0dB):        ');
         if (isempty(snr))
            snr = 0;
         end

         snrConst = 10^(snr/10);

         for k = 1:num_class
            cont = 1;
            classVar = [];
            varPower = num_features/snrConst;
            while(cont)
               classVar = rand(num_features-1,1)/snrConst;
               lastVar = varPower - sum(classVar);
               if (lastVar >= 0)
                  classVar = [classVar' lastVar]';
                  cont = 0;
               end
            end
            class_var = [class_var classVar];
         end
      else
         class_var = zeros(num_features,num_class);
      end
      % NOTE:  with class_mean and class_var, construct data then
      %        covariance matrix
   end
   class_mean
   class_var

   %*************************************************************************************
   % GENERATE class training/testing data
   % NOTE:  genclass_compare GENERATES/RETURNS training data and STORES
   %         testing realizations in work\test
   %        dim(classData) = num_features*num_class x num_data

   [classData,class_cov] = genclass_compare(num_data,class_mean,class_var);

   [rowData,num_data] = size(classData);
   if (rowData ~= num_features*num_class)
      disp('ERROR in data field')
   end

end
%****************************************************************************************
% NORMALIZE training and testing data by standard deviation (Method2)
[classData_norm] = dataMethod2(classData,class_mean,class_var);

%****************************************************************************************
% PLOT performance parameter and error surfaces/contours over a range
% of w and b
plotMS(num_class,num_features,classData,classData_norm)

%****************************************************************************************
% SET NN training parameters
a1=20;          % epochs between updating display
a2=500;         % maximum number of epochs to train
a3=100;         % initial learning rate
```

```
a4=2;          % learning rate increase
a5=0.5;        % learning rate decrease
a6=0.9;        % momentum constant
a7=1.04;       % maximum error ratio
tp = [a1 a2 a3 a4 a5 a6 a7];

% INITIALIZE/GENERATE 5 sets of weight and bias values.
w = 2*randn(num_features,5)-1;
b = randn(1,5);

% MONITOR MD, weight, bias update
%checkWB = [];
%save checkWB.dat checkWB -ascii -tabs

% INITIALIZE confusion matrix counters
% note:  reset confusion matrix when change class number, feature number, or SNR
reset = input('Reset confusion matrix counters (Y/N):       ','s');
if (reset == 'Y')
   typeA = zeros(num_class+1,num_class);
   typeB = zeros(num_class+1,num_class);
   typeB1 = zeros(num_class+1,num_class);
   typeC = zeros(num_class+1,num_class);
   typeStat = zeros(num_class+1,num_class);

   save typeA.dat typeA -ascii -tabs
   save typeB.dat typeB -ascii -tabs
   save typeB1.dat typeB1 -ascii -tabs
   save typeC.dat typeC -ascii -tabs
   save typeStat.dat typeStat -ascii -tabs
end

%*********************************************************************************************
%*********************************************************************************************
% A.  TRAIN/TEST standard MSNN
cd Method_SP1

disp(' ')
disp('****************************************************')
disp('A.  MSNN')
fig = 2000;

% type is CONFUSION MATRIX
% note:  type tracks confusion matrix for these 5-runs
%        typeA tracks confusion matrix for multiple 5-run
%        individual runs tracked by confusion matrix in simmsnn.m (i.e., type1)
type = zeros(num_class+1,num_class);
save type.dat type -ascii -tabs

for m = 1:5
   disp(['Run ',num2str(m)])
   simmsnn('trms_sp',1,classData,num_features,w(:,m),b(1,m),tp,fig);
   disp(' ')
   fig = fig+1+sum(1:(num_class-1));
end

load type.dat
disp(' ')
for m = 1:num_class+1
   disp(['TYPE',num2str(m),': ',num2str(type(m,:))])
end

cd ..
load typeA.dat
[Arow,Acol] = size(typeA);
tempA = typeA(Arow-num_class:Arow,:);
tempA = tempA + type;
typeA = [typeA;tempA];
```

```
save typeA.dat typeA -ascii -tabs

%*********************************************************************************************
%*********************************************************************************************
% B.  TRAIN/TEST  MSNN with normalized projection space (MSNN Mod 2)
cd Method_SP5

disp(' ')
disp('*****************************************************')
disp('B.  MSNN with norm projection space (MSNN Mod 3)')
fig = 2500;

% type is CONFUSION MATRIX
type = zeros(num_class+1,num_class);
save type.dat type -ascii -tabs

for m = 1:5
   disp(['Run ',num2str(m)])
   simmsnn('trms_sp5',5,classData,num_features,w(:,m),b(1,m),tp,fig);
   disp(' ')
   fig = fig+1+sum(1:(num_class-1));
end

load type.dat
disp(' ')
for m = 1:num_class+1
   disp(['TYPE',num2str(m),': ',num2str(type(m,:))])
end

cd ..
load typeB.dat
[Brow,Bcol] = size(typeB);
tempB = typeB(Brow-num_class:Brow,:);
tempB = tempB + type;
typeB = [typeB;tempB];
save typeB.dat typeB -ascii -tabs

%*********************************************************************************************
%*********************************************************************************************
% B1.  TRAIN/TEST MSNN and VMR termination reqmt (MSNN Mod 3)
cd Method_SP8

disp(' ')
disp('*****************************************************')
disp('B1.  MSNN with VMR termination (MSNN Mod 3)')
fig = 2500;

% type is CONFUSION MATRIX
type = zeros(num_class+1,num_class);
save type.dat type -ascii -tabs

for m = 1:5
   disp(['Run ',num2str(m)])
   simmsnn('trms_sp8',8,classData,num_features,w(:,m),b(1,m),tp,fig);
   disp(' ')
   fig = fig+1+sum(1:(num_class-1));
end

load type.dat
disp(' ')
for m = 1:num_class+1
   disp(['TYPE',num2str(m),': ',num2str(type(m,:))])
end

cd ..
load typeB1.dat
[B1row,B1col] = size(typeB1);
```

```
tempB1 = typeB1(B1row-num_class:B1row,:);
tempB1 = tempB1 + type;
typeB1 = [typeB1;tempB1];
save typeB1.dat typeB1 -ascii -tabs

%**************************************************************************************
%**************************************************************************************
% C.  TRAIN/TEST MSNN with preconditioned input space (MSNN Mod 1)
cd Method_SP2

disp(' ')
disp('****************************************************')
disp('C.  MSNN with precond input (MSNN Mod 1)')
fig = 3000;

% type is CONFUSION MATRIX
type = zeros(num_class+1,num_class);
save type.dat type -ascii -tabs

for m = 1:5
   disp(['Run ',num2str(m)])
   simmsnn_C(classData_norm,num_features,w(:,m),b(1,m),tp,fig);
   disp(' ')
   fig = fig+1+sum(1:(num_class-1));
end

load type.dat
disp(' ')
for m = 1:num_class+1
   disp(['TYPE',num2str(m),': ',num2str(type(m,:))])
end

cd ..
load typeC.dat
[Crow,Ccol] = size(typeC);
tempC = typeC(Crow-num_class:Crow,:);
tempC = tempC + type;
typeC = [typeC;tempC];
save typeC.dat typeC -ascii -tabs

%**************************************************************************************
%**************************************************************************************
% D.  PERCEPTRON NN
disp(' ')
disp('****************************************************')
disp('D.  Perceptron')
cd Method_SP7

% type is CONFUSION MATRIX
type = zeros(num_class+1,num_class);
save type.dat type -ascii -tabs
noType = 0;
save noType.dat noType -ascii -tabs

for m = 1:5
   disp(['Run ',num2str(m)])
   percptrnClassifier(num_class,classData,w(:,m),b(:,m))
   disp(' ')
end

load type.dat
for m = 1:num_class+1
   disp(['TYPE',num2str(m),': ',num2str(type(m,:))])
end
load noType.dat
disp(['BAD TYPE: ',num2str(noType)])
```

```
cd ..
load typeD.dat
[Drow,Dcol] = size(typeD);
tempD = typeD(Drow-num_class:Drow,:);
tempD = tempD + type;
typeD = [typeD;tempD];
save typeD.dat typeD -ascii -tabs

%*****************************************************************************************
%*****************************************************************************************
%% E. TEST iaw Brunzell/Eriksson quadratic classifier
disp(' ')
disp('****************************************************')
disp('E.  Statistical Classifier')

statClassifier(num_data,num_class,num_features,class_mean,class_cov)
```

2. Feature Simulation

a. userData.m

```
function [num_data,num_class,num_features,class_mean,class_var]...
       = userData(num_data,num_class,num_features,class_mean,class_var)

%*****************************************************************************************
% Function
%  - PROMPTS user for data specifications
%  - if no user data entered, default values used
%
% Use: [num_data,num_class,num_features,class_mean,class_var]
%      = userData(num_data,num_class,num_features,class_mean,class_var)
%
% Input/Returns
%    num_data:         number of training signals to construct
%    num_class:        number of signal classes
%    num_features:     number of distinguishing features
%    class_mean:       'num_class' 'num_features'x1 vectors of class feature means
%    class_var:        'num_class' 'num_features'x1 vectors of class feature variances
%
% 25 January 2000
% Miguel G. San Pedro
%*****************************************************************************************

disp('When asked for values, hit <enter> to use default values')

disp(' ')
num_data = input('Enter number of training signals (default=100):   ');
if (isempty(num_data))
   num_data = 100;
end

disp(' ')
num_class = input('Enter number of classes (default=3):      ');
if (isempty(num_class))
   num_class = 3;
end

disp(' ')
num_features = input('Enter number of features (default=3): ');
if (isempty(num_features))
   num_features = 3;
end
if (num_features < num_class)
   disp('ERROR:  number of distinguishing features > number of classes')
end
```

```
disp(' ')
userData = input('Enter mean for each feature for all classes (Y/N):        ','s');
if (userData == 'Y')
   for k = 1:num_class
      getData = input(['Enter mean for class',num2str(k),...
            ' features (enter as column vector):    ']);
      [rowData,colData] = size(getData);
      if(rowData*colData ~= num_features)
         disp('*** DATA ENTRY ERROR ***')
      else
         if (colData ~=1)
            getData = reshape(getData,rowData*colData,1);
         end
      end
      class_mean(:,k) = getData;
   end
else
   class_mean = 2*rand(num_features,num_class) - 1;
end

disp(' ')
userData = input('Enter variance for each feature for all classes (Y/N):   ','s');
if (userData == 'Y')
   for k = 1:num_class
      getData = input(['Enter variance for class',num2str(k),...
            ' features (enter as column vector):    ']);
      [rowData,colData] = size(getData);
      if(rowData*colData ~= num_features)
         disp('*** DATA ENTRY ERROR ***')
      else
         if (colData ~=1)
            getData = reshape(getData,rowData*colData,1);
         end
      end
      class_var(:,k) = getData;
   end
else
   % Randomly DETERMINE variance and ADD white noise
   snr = [];
   class_var = [];
   usrSNR = input('Add noise (Y/N):   ','s');
   if (usrSNR == 'Y')
      snr = input('Enter feature SNR (default=0dB): ');
      if (isempty(snr))
         snr = 0;
      end

      snrConst = 10^(snr/10);

      for k = 1:num_class
         cont = 1;
         classVar = [];
         varPower = num_features/snrConst;
         while(cont)
            classVar = rand(num_features-1,1)/snrConst;
            lastVar = varPower - sum(classVar);
            if (lastVar >= 0)
               classVar = [classVar' lastVar]';
               cont = 0;
            end
         end
         class_var = [class_var classVar];
      end
   else
      class_var = zeros(num_features,num_class);
   end
end
```

```
% NOTE:  with class_mean and class_var, construct data then covariance matrix

return
```

b. genclass_compare.m

```
function [difclass,class_cov] = genclass_compare(numData,class_mean,class_var);

%***************************************************************************************
% Function
%  - Randomly GENERATES 'numData' training realizations of 'num_class' classes (note:
%    num_class plotting limited to <= 5).
%  - CALCULATES covariance matrix of data for statistical analysis
%  - PRE-CONDITIONS class data for use by Method2 by normalizing data by standard
%    deviation, resulting in "testcl#" data (normalized data vice normalized
%    projections).
%  - GENERATES 10*'numData' test realizations.
%
% Use: [classdata,class_cov] = genclass_compare(numData,num_class,class_mean,class_var);
%
% Input    numData:    number of training signals to construct
%          class_mean: 'num_class' 'num_features'x1 vectors of class feature means
%          class_var:  'num_class' 'num_features'x1 vectors of class feature variances
%
% Returns  difclass:   generated training data points
%          class_cov:  'num_class' 'num_features'x'num_features' covariance matrix
%
% Saves at directory test/, testing realizations
%
% 14 January 2000
% Miguel G. San Pedro
%***************************************************************************************
plot_char = ['b*';'r+';'go';'cs';'md'];
class_cov = [];
difclass = [];

% TRAINING REALIZATIONS
figure(1)
orient tall

[num_features,num_class] = size(class_mean);
% GENERATE numData training realizations
for m = 1:num_class
   classData = sqrt(class_var(:,[m*ones(1,numData)])).*randn(num_features,numData)...
              + class_mean(:,[m*ones(1,numData)]);
   class_cov = [class_cov cov(classData')];
   difclass = [difclass;classData];

   % PLOT first three features of each class
   subplot(211)
   plot3(classData(1,:),classData(2,:),classData(3,:),plot_char(m,:))
   hold on
   xlabel('First Feature');
   ylabel('Second Feature');
   zlabel('Third Feature');
   title('Training Data')
   box on
   grid on

   subplot(234)
   plot(classData(1,:),classData(2,:),plot_char(m,:))
   hold on
   xlabel('First Feature');
   ylabel('Second Feature');
   grid on
```

```
    subplot(235)
    plot(classData(1,:),classData(3,:),plot_char(m,:))
    hold on
    xlabel('First Feature');
    ylabel('Third Feature');
    grid on

    subplot(236)
    plot(classData(2,:),classData(3,:),plot_char(m,:))
    hold on
    xlabel('Second Feature');
    ylabel('Third Feature');
    grid on
end
hold off

%****************************************************************************************
% GENERATE
%   - numData*10 test realizations of each classes
%   - test_data realizations of random noise that should not type to any classes
test_data = numData*10;
testClass = [];

for k = 1:num_class
   cl_SD = [];
   cl_SD = sqrt(class_var(:,[k*ones(1,test_data)]));
   cl_Mean = [];
   cl_Mean = class_mean(:,[k*ones(1,test_data)]);

   trainData = cl_SD.*randn(num_features,test_data) + cl_Mean;
   testClass = [testClass;trainData];
end

% GENERATE non-class data for testing
nonClassData = 10*randn(num_features,test_data) - 5;
testClass = [testClass;nonClassData];
save test\testClass.dat testClass -ascii -tabs

return
```

3. Modulated Signal Simulation and Feature Extraction

a. genSignal.m

```
function [featuresSave,meanSig,covSig,varSig]...
                      = genSignal(fxn,num_signals,A,T,f,n,features,snr)

%****************************************************************************************
% Function
%  - GENERATES training and testing signals
%
% Use: [featuresSave,meanSig,covSig,varSig]
%                     = genSignal(fxn,num_signals,A,T,f,n,features,snr)
%
% Input    fxn:               string name of signal type to construct
%                             ('2-ASK', '2-PSK', or '2-FSK')
%          num_signals:       number of training signals to construct; constructs
%                             10*num_signals testing signals
%          A:                 signal amplitude
%          T:                 signal period
%          f:                 carrier frequency
%          n:                 time sample vector
%          features:          distinguishing features indices (from detFeatures.m)
%          snr:               signal SNR
%
% Returns  featuresSave:      distinguishing features extracted for classifying
```

```
%         meanSig:            mean of extracted features
%         covSig:             covariance matrix of extracted features
%         varSig:             variance of extracted features
%
% 31 January 2000
% Miguel G. San Pedro
%*****************************************************************************************

% GENERATE training signals
featuresSave = [];
for k = 1:num_signals
   [signal,featuresSignal] = feval(fxn,A,T,f,n,features,snr);
   featuresSave = [featuresSave featuresSignal];
end
meanSig = mean(featuresSave,2);
covSig = cov(featuresSave');
[covSigRow,covSigCol] = size(covSig);
for k = 1:covSigRow
   for kk = 1:covSigCol
      if (~covSig(k,kk))             % element is zero
         covSig(k,kk) = 1e-10;
      end
   end
end
varSig = diag(covSig);
%goon = input('continue ','s');
%if goon == 'y'
%   varSig
%   meanSig
%end

% GENERATE testing signals
load testClass.dat
testClassSave = [];
for k = 1:10*num_signals
   [signal,testSignal] = feval(fxn,A,T,f,n,features,snr);
   testClassSave = [testClassSave testSignal];
end
testClass = [testClass;testClassSave];
save test\testClass.dat testClass -ascii -tabs

return
```

b. gen2ASK.m, gen2PSK.m, gen2FSK.m

```
function [signal,features2ASK] = gen2ASK(A,T,fc,n,features,snr)

%*****************************************************************************************
% Function
%  - GENERATES a 2ASK signal
%
% Use: [signal,features2ASK] = gen2ASK(A,T,fc,n,features,snr)
%
% Input    A:                 signal amplitude
%          T:                 bit period
%          fc:                carrier frequency
%          n:                 time sample vector
%          features:          distinguishing features indices (from detFeatures.m)
%          snr:               signal SNR
%
% Returns  signal:            postive frequencies of Fourier transformed 2-ASK signal
%                             realization
%         features2ASK:       distinguishing features spectral magnitudes
%
% 21 January 2000
% Miguel G. San Pedro
```

```
%*****************************************************************************************

% GENERATE message
a = zeros(1,20);
while (sum(a) == 0)
   a = round(rand(1,20));
end
basis = A/sqrt(T)*sin(2*pi*fc*n);                    % SET basis function

msg = [];
for kk = 1:length(a)
   msg = [msg a(kk)*basis];
end
[msgRow,msgCol] = size(msg);
v = reshape(msg,1,msgRow*msgCol);

% ADD white noise
if ((nargin >=5) & (snr ~= 9999))
   energyV = v*v';
   varNoise = (energyV/length(n))/10^(snr/10);
   noise = sqrt(varNoise)*randn(size(v));
   v = v + noise;
end

% NORMALIZE the signal power
den = v*v';
v = v/sqrt(den);

% PRE-PROCESS signal
% - use decision rule to extract points
[sigRow,sigCol] = size(v);
iter = floor(sigCol/250);                            % discard leftover points
aveSig = zeros(1,1000);
for k = 1:iter
   % FFT signal
   block = v(1,250*k-249:250*k);
   sigFFT = abs(fft(block,1000));
   aveSig = aveSig + sigFFT;
end
signal = aveSig(1:length(aveSig)/2)/iter;

features2ASK = [];
if (nargin >= 5)
   features2ASK = signal(features)';
end

return
```

```
function [signal,features2PSK] = gen2PSK(A,T,fc,n,features,snr)

%*****************************************************************************************
% Function
%   - GENERATES a 2PSK signal
%
% Use: [signal,features2PSK] = gen2PSK(A,T,fc,n,features,snr)
%
% Input     A:                  signal amplitude
%           T:                  bit period
%           fc:                 carrier frequency
%           n:                  time sample vector
%           features:           distinguishing features indices (from detFeatures.m)
%           snr:                signal SNR
%
% Returns   signal:             postive frequencies of Fourier transformed 2-PSK signal
%                               realization
%           features2PSK:       distinguishing features spectral magnitudes
```

```
%
% 21 January 2000
% Miguel G. San Pedro
%**************************************************************************************

% GENERATE message
a = 2*round(rand(1,20)) - 1;
basis = A*sqrt(2/T)*sin(2*pi*fc*n);          % SET basis function

msg = [];
for kk = 1:length(a)
   msg = [msg a(kk)*basis];
end
[msgRow,msgCol] = size(msg);
msg = reshape(msg,1,msgRow*msgCol);

v = msg;

% ADD white noise
if ((nargin >=5) & (snr ~= 9999))
   energyV = v*v';
   varNoise = (energyV/length(n))/10^(snr/10);
   noise = sqrt(varNoise)*randn(size(v));
   v = v + noise;
end

% NORMALIZE the signal power
v = v/sqrt(v*v');

% PRE-PROCESS signal
% - use decision rule to extract points
[sigRow,sigCol] = size(v);
iter = floor(sigCol/250);                    % discard leftover points
aveSig = zeros(1,1000);
for k = 1:iter
   % FFT signal
   block = v(1,250*k-249:250*k);
   sigFFT = abs(fft(block,1000));
   aveSig = aveSig + sigFFT;
end
signal = aveSig(1:length(aveSig)/2)/iter;

features2PSK = [];
if (nargin >= 5)
   features2PSK = signal(features)';
end

return
```

```
function [signal,features2FSK] = gen2FSK(A,T,fc,n,features,snr)

%**************************************************************************************
% Function
%  - GENERATES a 2FSK signal
%
% Use: [signal,features2FSK] = gen2FSK(A,T,fc,n,features,snr)
%
% Input    A:              signal amplitude
%          T:              bit period
%          fc:             carrier frequency
%          n:              time sample vector
%          features:       distinguishing features indices (from detFeatures.m)
%          snr:            signal SNR
%
% Returns  signal:         postive frequencies of Fourier transformed 2-FSK signal
%                          realization
```

```
%           features2FSK:       distinguishing features spectral magnitudes
%
% 21 January 2000
% Miguel G. San Pedro
%*****************************************************************************************
delf = 1/T;

% GENERATE message
a = round(rand(1,20));

basis = [];
for kk = 1:length(a)
   if (a(kk) == 1)
      basis = [basis sqrt(2/T)*sin(2*pi*fc*n)];
   else
      basis = [basis sqrt(2/T)*sin(2*pi*(fc+delf)*n)];
   end
end
msg = basis;
[msgRow,msgCol] = size(msg);
msg = reshape(msg,1,msgRow*msgCol);

v = A*msg;

% ADD white noise
if ((nargin >=5) & (snr ~= 9999))
   energyV = v*v';
   varNoise = (energyV/length(n))/10^(snr/10);
   noise = sqrt(varNoise)*randn(size(v));
   v = v + noise;
end

% NORMALIZE the signal power
v = v/sqrt(v*v');

% PRE-PROCESS signal
% - use decision rule to extract points
[sigRow,sigCol] = size(v);
iter = floor(sigCol/250);                          % discard leftover points
aveSig = zeros(1,1000);
for k = 1:iter
   % FFT signal
   block = v(1,250*k-249:250*k);
   sigFFT = abs(fft(block,1000));
   aveSig = aveSig + sigFFT;
end
signal = aveSig(1:length(aveSig)/2)/iter;

features2FSK = [];
if (nargin >= 5)
   features2FSK = signal(features)';
end

return
```

c. *detFeatures.m, extractFeatures.m*

```
function [features] = detFeatures

%*****************************************************************************************
% Function
%  - EXTRACTS feature indices to be used for signal classification
%
% Use: [featuresLoc] = extractFeatures(sigType,signal)
%
% Input     (none)
```

```
%
% Returns  features:  signal component indices for signal classification
%
% 21 January 2000
% Miguel G. San Pedro
%**************************************************************************************************
clear

A = 4;                    % SET signal amplitude
T = 1e-6;                 % SET bit interval of signal (sec)
fs = 5e7;                 % SET bit sampling frequency (samples/sec)
fc = 5e6;                 % SET carrier frequency (Hz)
n = linspace(0,T,fs*T);
features = [];

% DETERMINE class1 features:  2-ASK
featuresSave = [];
for k = 1:1000
   [ASK,temp] = gen2ASK(A,T,fc,n);
   featuresLoc = extractFeatures('2ASK',ASK);
   if (k ~= 1)
      featuresSave = intersect(featuresSave,featuresLoc);
   else
      featuresSave = featuresLoc;
   end
end
features2ASK = featuresSave;
disp(size(features2ASK))

features = union(features,features2ASK);

% DETERMINE class2 features:  2-PSK
featuresSave = [];
for k = 1:1000
   [PSK,temp] = gen2PSK(A,T,fc,n);
   featuresLoc = extractFeatures('2PSK',PSK);
   if (k ~= 1)
      featuresSave = intersect(featuresSave,featuresLoc);
   else
      featuresSave = featuresLoc;
   end
end
features2PSK = featuresSave;
disp(size(features2PSK))

features = union(features,features2PSK);

% DETERMINE class3 features:  2-FSK
featuresSave = [];
for k = 1:1000
   [FSK,temp] = gen2FSK(A,T,fc,n);
   featuresLoc = extractFeatures('2FSK',FSK);
   if (k ~= 1)
      featuresSave = intersect(featuresSave,featuresLoc);
   else
      featuresSave = featuresLoc;
   end
end
features2FSK = featuresSave;
disp(size(features2FSK))

features = union(features,features2FSK);

return
```

```
function [featuresLoc] = extractFeatures(sigType,signal)

%*************************************************************************************
% Function
%  - EXTRACTS feature indices satisfying prescribed decision rule
%
% Use: [featuresLoc] = extractFeatures(sigType,signal)
%
% Input    sigType:          string specifying signal type
%           signal:          signal frequency components
%
% Returns  featuresLoc:      indices of signal components satisfying prescribed decision
%                            rule
%
% 21 January 2000
% Miguel G. San Pedro
%*************************************************************************************

npoints = 2;                  % npoints specifies feature spacing
featuresLoc = 30:npoints:130; % decision rule

return
```

4. Data Conditioning and Display

a. dataMethod2.m

```
function [classData_norm] = dataMethod2(classData,class_mean,class_var)

%*************************************************************************************
% Function
%  - NORMALIZES training and testing data by class standard deviation for use in Method2
%
% Use: [classData_norm] = dataMethod2(classData,class_mean,class_var)
%
% Input    classData:        generated training data
%          class_mean:       'num_class' 'num_features'x1 vectors of class feature means
%          class_var:        'num_class' 'num_features'x1 vectors of class feature
%                            variances
%
% Returns  classData_norm:   normalized training data
%
% Saves at directory test/, normalized testing realizations
%
% 14 January 2000
% Miguel G. San Pedro
%*************************************************************************************

classData_norm = [];
[num_features,num_class] = size(class_mean);
[rowData,num_data] = size(classData);

% NORMALIZE training data by standard deviation (Method2)
if (num_features*num_class ~= rowData)
   disp('Note:  INPUT ERROR')
else
   for k = 1:num_class
      knum_feat = k*num_features;
      data = classData(knum_feat - num_features + 1:knum_feat,:);
      data_adj = (data - class_mean(:,[k*ones(1,num_data)]))...
                   ./sqrt(class_var(:,[k*ones(1,num_data)]))...
                   + class_mean(:,[k*ones(1,num_data)]);
      classData_norm = [classData_norm;data_adj];
   end
end
```

```
% NORMALIZE testing data by standard deviation (Method2)
testClass_norm = [];
load test\testClass.dat
[rowData,num_test] = size(testClass);

if (num_features*(num_class+1) ~= rowData)
   disp('Note:  INPUT ERROR')
else
   for k = 1:num_class+1
      knum_feat = k*num_features;
      data = testClass(knum_feat - num_features + 1:knum_feat,:);
      data_adj_save = [];
      for kk = 1:num_class
         data_adj = (data - class_mean(:,[kk*ones(1,num_test)]))...
                    ./sqrt(class_var(:,[kk*ones(1,num_test)]))...
                    + class_mean(:,[kk*ones(1,num_test)]);
         data_adj_save = [data_adj_save;data_adj];
      end
      testClass_norm = [testClass_norm data_adj_save];
   end
end

save test\testClass_norm.dat testClass_norm -ascii -tabs

return
```

b. plotMS.m, errsurf_sp.m

```
function plotMS(num_class,num_features,classData,classData_norm)

%*****************************************************************************************
% Function
%    PLOTS projection of test data using weights and bias determined by the mean
%    separator neural network
%
% Use: plotMS(num_class,num_features,classData,classData_norm)
%
% Input    num_class:          number of signal classes
%          num_features:       number of distinguishing features
%          classData:          class data training set
%          classData:          class data training set (normalized - Method2)
%
% Limitations:                  can plot only 1 feature classes
%
% Returns (none)
%
% 12 January 2000
% Miguel G. San Pedro
%*****************************************************************************************
global gloUsrReq

w1 = [];
if (gloUsrReq == 'N')
   userReq = input('Plot Mean Separator and Error surface and contours (Y/N):      ','s');
   if (userReq == 'Y')
      f = ['meansep_sp1';'meansep_sp2';'meansep_sp3';'meansep_sp5'];
      w1 = input('Enter weight/bias range (default -100:100):      ');
      b1 = w1;
      if (isempty(w1))
         w1 = [-50:.25:50];
         b1 = w1;
      end
      for k = 1:4
         for m = 1:num_class
            mnum_feat = m*num_features;
```

```
            for mm = m+1:num_class
               mmnum_feat = mm*num_features;
               if (k ~= 2)
                  cl1 = classData(mnum_feat - num_features + 1:mnum_feat,:);
                  cl2 = classData(mmnum_feat - num_features + 1:mmnum_feat,:);
                  p = [cl1;cl2];
               else
                  cl1 = classData_norm(mnum_feat - num_features + 1:mnum_feat,:);
                  cl2 = classData_norm(mmnum_feat - num_features + 1:mmnum_feat,:);
                  p = [cl1;cl2];
               end
               errsurf_sp(p,w1,b1,f(k,:));
            end
         end
      end
   end
end

return
```

```
function m = errsurf_sp(p,wv,bv,f)

%******************************************************************************************
% Function
%    PLOTS the error surface and error contours of a mean seperator neural network over a
%    range of weights and biases
%
% Use      m = errmesh_sp(p,wv,bv,f)
%
% Input     p:  2xQ matrix of input vectors.  First row - feature of class 1; second row -
%               feature of class 2 in second row
%          wv:  column vector of weights
%          bv:  column vector biases
%           f:  transfer function (optional. default - meansep_sp5)
%
% Returns  m:  matrix of error values over wv and bv.
%
% Example
%       p = [-6.0 -6.1 -4.1 -4.0 +4.0 +4.1 +6.0 +6.1;
%            +0.0 +0.0 +.97 +.99 +.01 +.03 +1.0 +1.0];
%       wv = (-1:.1:1)';
%       bv = (-2.5:.25:2.5)';
%       es = errmesh_sp(p,wv,bv,'meansep_sp5');
%
% 5 January 2000
% Miguel G. San Pedro
%******************************************************************************************

if nargin < 3,error('Not enough input arguments.');end
if (nargin == 3)
   f = 'meansep_sp5';
end

[pRow,pCol] = size(p);
p1 = p(1,:);
p2 = p(2,:);

if (f == 'meansep_sp1')
   t = -400;
end
if (f == 'meansep_sp2')        % for meansep_sp2, refer to notes in meansep_sp2 function
                               % code
   t = -400;
   f = 'meansep_sp1';
end
if (f == 'meansep_sp5')        % for MSNN norm proj var, no identifiable optimum value.
```

```
                                % Algorithm is such that want to increase mean spread and
                                % decrease sum of variance.  Result wanted is large
                                % magnitude for value of performance parameter.  Therefore,
                                % set t=0 ==> error plot and performance plot are the same.
   t = 0;
end

m = zeros(length(bv),length(wv));
for k = 1:length(wv)
   for kk = 1:length(bv)
      pp(kk,k) = feval(f,p1,p2,wv(k),bv(kk));
      if (f == 'meansep_sp3')
         if (pp(kk,k) <= 400)
            t = 0;
         else
            t = 1600;
         end
      end
      m(kk,k) = (t - pp(kk,k))^2;  % squared error calculation
   end
end

% PLOT performance parameter suface and contours
figure
orient landscape
subplot(221)
grid
mesh(bv,wv,pp)
xlabel('bias')
ylabel('weight')
zlabel('Mean Separator')
title(['Performance Parameter Surface (',f,')'])

subplot(222)
grid
contour(bv,wv,pp,10)
xlabel('bias')
ylabel('weight')
title(['Performance Parameter Contours (',f,')'])

% PLOT error surface and contours
subplot(223)
grid
mesh(bv,wv,m)
xlabel('bias')
ylabel('weight')
zlabel('error')
title(['Error Surface (',f,')'])

subplot(224)
grid
contour(bv,wv,m,10)
xlabel('bias')
ylabel('weight')
title(['Error Contours (',f,')'])

return
```

c. dispProjection.m, plotProjection.m, dispWeightBias.m

```
function dispProjection(o,r,numTestPts,method)

%*****************************************************************************************
% Function
%    DISPLAYS the projection of test data using weights and bias determined by the mean
%    separator neural network

```

```
%
% Use: dispProjection(o,r,numTestPts,method)
%
% Input    o:          matrix of all test data projection
%          r:          matrix of class identification projection
%          numTestPts: number of test data points
%          method:     method number
%
% Returns (none)
%
% 12 January 2000
% Miguel G. San Pedro
%**************************************************************************************

% DISPLAY class type identifiers and testing data projection (considers each class
% separately)

[n,all_classes] = size(o);
num_class = all_classes/numTestPts - 1;      % -1 so do not count noise block as a
                                             % distinct class
for k = 1:num_class
   knumTestPts = k*numTestPts;
   data = o(:,knumTestPts - numTestPts + 1:knumTestPts);

   disp(['r',num2str(k),' = ',num2str(r(:,k)')])
   disp(['o',num2str(k),' = '])
   disp(num2str(data'))
   disp(' ')
end

return
```

```
function plotProjection(o,r,numTestPts,method,fig)

%**************************************************************************************
% Function
%    PLOTS projection of test data using weights and bias determined by the mean
%    separator neural network
%
% Use: plotProjection(o,r,numTestPts,method,fig)
%
% Input    o:          matrix of all test data projection
%          r:          matrix of class identification projection
%          numTestPts: number of test data points
%          method:     method number
%          fig:        figure number
%
% Limitations: - o and r can only contain 3 rows of data
%              - only 5 classes can be plotted
%
% Returns (none)
%
% 12 January 2000
% Miguel G. San Pedro
%**************************************************************************************

[n,all_classes] = size(o);
num_class = all_classes/numTestPts - 1;      % -1 to discount noise block as a distinct
                                             % class
% limit number of classes to plot to 5
if (num_class > 5)
   num_class = 5;
end
plot_char = ['b*';'r+';'go';'cs';'md'];

figure(fig)
```

```
orient tall
for k = 1:num_class
   % considers each class separately
   knumTestPts = k*numTestPts;
   data = o(:,knumTestPts - numTestPts + 1:knumTestPts);

   subplot(211)
   plot3(data(1,1:5:length(data)),data(2,1:5:length(data)),data(3,1:5:length(data)),
         plot_char(k,:))
   hold on
   plot3(r(1,k),r(2,k),r(3,k),plot_char(k,:))

   subplot(234)
   plot(data(1,1:5:length(data)),data(2,1:5:length(data)),plot_char(k,:))
   hold on
   plot(r(1,k),r(2,k),plot_char(k,:))

   subplot(235)
   plot(data(2,1:5:length(data)),data(3,1:5:length(data)),plot_char(k,:))
   hold on
   plot(r(2,k),r(3,k),plot_char(k,:))

   subplot(236)
   plot(data(1,1:5:length(data)),data(3,1:5:length(data)),'b*')
   hold on
   plot(r(1,k),r(3,k),plot_char(k,:))
end

subplot(211)
title(['Test Data Projection (Method',num2str(method),')'])
xlabel('feature 1')
ylabel('feature 2')
zlabel('feature 3')
box on
grid on
hold off

subplot(234)
grid on
xlabel('feature 1')
ylabel('feature 2')
hold off

subplot(235)
grid on
xlabel('feature 2')
ylabel('feature 3')
hold off

subplot(236)
grid on
xlabel('feature 1')
ylabel('feature 3')
hold off

return
```

```
function dispWeightBias(w,b)

%*****************************************************************************************
% Function
%    DISPLAYS weights and biases determined during training phase
%
% Use:  dispWeightBias(w,b)
%
% Input    w:          projection weight vector
```

```
%          b:          projection bias
%
% Returns (none)
%
% 27 December 1999
% Miguel G. San Pedro
%*******************************************************************************************

[num_prwise,num_class] = size(w);

% DISPLAY weights/bias and class type identifiers
for k = 1:num_class
   disp(['wNN',num2str(k),' = [',num2str(w(:,k)'),'] bNN',num2str(k), ' = ',...
          num2str(b(k))]);
   disp(' ')
end

return
```

B. CLASSIFICATION METHODS

This section contains the programs used to determine the classification capability of the specific signal typing methods.

1. Statistical Classifier

a. statClassifier.m

```
function statClassifier(num_data,num_class,num_features,...
                                      class_mean,class_cov)

%*******************************************************************************************
% Function
%    USES quadratic classifier to type classes
%
% Use: statClassifier(num_data,num_class,num_features,class_mean,class_cov)
%
% Input     num_data:          number of training realizations
%           num_class:         number of signal classes
%           num_features:      number of distinguishing features
%           class_mean:        feature mean values
%           class_cov:         feature covariance matrix
%
% Returns (none)
%
% 7 March 2000
% Miguel G. San Pedro
%*******************************************************************************************

% LOAD test points
load test\testClass.dat

[testRow,testData] = size(testClass);
if (10*num_data ~= testData)
   disp('*** DATA ERROR ***')
end

% SET class a priori probabilities for equiprobably classes
P = 1/num_class;

% LOAD stat classifier confusion matrix
load typeStat.dat
```

```
data = [];
tempMat = [];
for k = 1:num_class
   knum_feat = k*num_features;
   data = testClass(knum_feat - num_features + 1:knum_feat,:);

   distMat = [];
   for kk = 1:num_class
      kknum_feat = kk*num_features;
      dist = classDist(data,P,class_mean(:,kk),...
               class_cov(:,kknum_feat - num_features + 1:kknum_feat));
      distMat = [distMat;dist];
   end

   type = zeros(1,num_class);
   for kk = 1:testData'
      [y index] = min(distMat(:,kk),[],1);
      type(index) = type(index) + 1;
   end

   disp(['TYPE',num2str(k),': ',num2str(type)])

   [Statrow,Statcol] = size(typeStat);
   tempStat = typeStat(Statrow-(num_class-k),:);
   tempStat = tempStat + type;
   tempMat = [tempMat;tempStat];
end

typeStat = [typeStat;tempMat];
save typeStat.dat typeStat -ascii -tabs

return
```

b. ClassDist.m

```
function [dist] = classDist(data,classProb,classMean,classCov)

%*******************************************************************************************
% Function
%  - DETERMINES classification distance for test data wrt to a particular class'
%    statistics (as discussed by Brunzell/Eriksson)
%  - distance parameter given by
%      di(x) = ln(det(classCov)) - 2*lnP + (x-classMean)'*inv(classCov)*(x-classMean)
%
% Use: [dist] = classDist(data,classProb,classMean,classCov)
%
% Input     data:       m-dimensional test data to be typed (m rows)
%           classProb:  class a priori probability
%           classMean:  mx1 vector of class feature mean values
%           classCov:   mxm covariance matrix for class features
%
% Returns   dist:       distance for each test data point
%
% 7 January 2000
% Miguel G. San Pedro
%*******************************************************************************************

[dataRow,dataCol] = size(data);

dist = [];
c1 = log(det(classCov)) - 2*log(classProb);
c2 = inv(classCov);
for k = 1:dataCol
   c3 = data(:,k) - classMean;
   dist(k) = c1 + c3'*c2*c3;
end
```

```
return
```

2. Perceptron

a. percptrnClassifier.m

```
function percptrnClassifier(num_class,snr,classData,w,b)

%*****************************************************************************************
% Function
%    USES quadratic classifier to type classes
%
% Use: percptrnClassifier(num_class,classData,w,b)
%
% Input    num_class: number of signal classes
           snr:       signal snr
%          classData: class data training set
%          w:         projection weight vector
%          b:         projection bias
%
% Returns  (none)
%
% 15 January 2000
% Miguel G. San Pedro
%*****************************************************************************************

[totFeatures,numData] = size(classData);
numFeatures = totFeatures/num_class;

% TRAINING PHASE
% ORGANIZE input/target vector
p = [];
t = [];
target = detTargVect(num_class);
for k = 1:num_class
   knumFeat = k*numFeatures;
   p = [p classData(knumFeat - numFeatures + 1:knumFeat,:)];
   t = [t target(:,[k*ones(1,numData)])];
end

[numNeurons,tCol] = size(t);
if (tCol ~= num_class*numData)
   disp('*** DATA ERROR')
end

net = newp(minmax(p),numNeurons,'hardlim','learnp');
w = w';
w = w([ones(1,numNeurons)],:);
net.iw{1,1} = w;
net.b{1} = b([ones(1,numNeurons)],:);
net.trainParam.epochs = 2500;
figure
[net,tr] = train(net,p,t);

disp('Final neuron weights and bias')
wNN = net.iw{1,1}
bNN = net.b{1}

maxEpoch = max(tr.epoch);
load snrEpoch.dat
snrEpoch = [snrEpoch; snr maxEpoch];
save snrEpoch.dat snrEpoch -ascii -tabs

load ..\test\testClass.dat
[testRow,numTestData] = size(testClass);
```

```
if(testRow ~= numFeatures*(num_class+1))
   disp('*** DATA ERROR')
end

% TESTING PHASE
% REORGANIZE testClass to place blocks of class test data in a row vice in a column
pTest = [];
for k = 1:num_class+1
   knumFeat = k*numFeatures;
   pTest = [pTest testClass(knumFeat - numFeatures + 1:knumFeat,:)];
end
tTest = sim(net,pTest);

% COUNT results
type1 = zeros(num_class+1,num_class);
noType1 = 0;
for k = 1:(num_class+1)*numTestData
   typeRow = ceil(k/numTestData);
   index = bi2de(flipud(tTest(:,k))');
   if ((index == 0)|(index > num_class))
      if (typeRow <= num_class)
         noType1 = noType1 + 1;   % do not count noType if random test data
      end
   else
      type1(typeRow,index) = type1(typeRow,index) + 1;
   end
end

% DISPLAY test data class typing
for m = 1:num_class+1
   disp(['type',num2str(m),': ',num2str(type1(m,:)),'       ',num2str(numTestData)])
end
disp(['no type: ',num2str(noType1)])
disp(' ')

load type.dat
type = type + type1;
save type.dat type -ascii -tabs

load noType.dat
noType = noType + noType1;
save noType.dat noType -ascii -tabs

return
```

b. detTargVect.m

```
function [target] = detTargVect(num_class)

%*******************************************************************************************
% Function
%    DETERMINES perceptron target vector
%
% Use: [target] = detTargVect(num_class)
%
% Input     num_class:  number of signal classes
%
% Returns   target:     vector of unique binary class representations
%
% Example:      num_class = 6;
%               [target] = detTargVect(num_class)
%               class = [1 2 3 4 5 6]
%               target = [0 0 0 1 1 1;
%                         0 1 1 0 0 1;
%                         1 0 1 0 1 0]
%
```

```
% 15 January 2000
% Miguel G. San Pedro
%*****************************************************************************************

class = [1:num_class];
[target] = flipud(de2bi(class)');

return
```

3. Common Mean Separator Programs

a. simmsnn.m

```
function simmsnn(f,method,classData,num_features,w,b,tp,fig)

%*****************************************************************************************
% Function
%    SIMULATES the mean separator neural network with performance parameter defined by
%    function f
%
% Use: simmsnn(f,method,classData,num_features,w,b,tp,fig)
%
% Input     f:          mean separator neural network function method
%           method:     mean separator variation number
%                       1 - standard
%                       2 - preconditionied input (Mod 1)
%                       5 - normalized projection (Mod 2)
%                       8 - with VMR termination (Mod 3)
%           classData:  training data
%           w:          projection weight vector
%           b:          projection bias
%           tp:         training parameters (see function trms_sp)
%           fig:        figure number
%
% Returns   (none)
%
% 6 March 2000
% Miguel G. San Pedro
%*****************************************************************************************
global gloUsrReq

[classRow,num_data] = size(classData);
num_class = classRow/num_features;
num_prwise = sum(1:num_class-1);      % number of pairwise comparisons
ind = 0;                              % pairwise index
r = zeros(num_prwise,num_class);      % class type identifier

%*****************************************************************************************
% COMPARE class k and class kk
for k = 1:num_class
   knum_feat = k*num_features;
   for kk = k+1:num_class
      kknum_feat = kk*num_features;
      ind = ind + 1;

      class1 = classData(knum_feat-num_features+1:knum_feat,:);
      class2 = classData(kknum_feat-num_features+1:kknum_feat,:);
      p1 = [class1;class2];

      disp(['Class ',num2str(k),' vs Class ',num2str(kk)])
      fig = fig+1;
      [wNN(:,ind),bNN(ind)] = feval(f,w,b,p1,tp,method,fig);

      % DETERMINE class type identifier for this pairwise comparison
      for mm = 1:num_class
         mmnum_feat = mm*num_features;
```

```
         classA = classData(mmnum_feat-num_features+1:mmnum_feat,:);
         r(ind,mm) = 20*mean(logsig(wNN(:,ind)'*classA + bNN(ind)))-10;

         % DETERMINE projection data for neuron maps
         plotr = [plotr 20*logsig(wNN(:,ind)'*classA + bNN(ind))-10];

      end

   % PLOT neuron maps
   figure
   plot(plotr)
   xlabel('Test Point')
   ylabel(['Neuron Map [',num2str(k),',',num2str(kk),']'])

   end
end

% DISPLAY weights/bias and class type identifiers
if (gloUsrReq == 'N')
   userReq = input('Display projection weights and biases (Y/N):     ','s');
   if (userReq == 'Y')
      dispWeightBias(wNN,bNN)
   end
   disp(' ')
end

%*******************************************************************************************
% CLASSIFY test points
load ..\test\testClass.dat
[testRow,testData] = size(testClass);
if (testRow ~= num_features*(num_class+1))
   disp('*** DATA ERROR')
end

% REORGANIZE test data into a matrix with dimensions
% 'num_features'x'num_class'*'num_data'
testCl = [];
for m = 1:num_class+1
   testCl = [testCl testClass((m-1)*num_features+1:m*num_features,:)];
end
[testRow,totTestData] = size(testCl);
if ((testRow ~= num_features)|(totTestData ~= (num_class+1)*testData))
   disp('*** DATA ERROR')
end

% PROJECT/TYPE testClass data
% 'diff' matrices store distances from class type identifiers (r's) to data projections
% (o's) determine best fit (i.e. trial data typing) by deteriming minimum value of each
% row
% 2nd dimension of r gives number of classes, testData gives number of test data points
% taking column number of each testProj point and performing ceil(colNum/testData) gives
% class number

testProj = [];
type1 = zeros(num_class+1,num_class);

if (gloUsrReq == 'N')
   userReq = input('Display typing distance data (Y/N):      ','s');
else
   userReq = 'N';
end
for m = 1:totTestData
   for mm = 1:num_prwise
      o(mm,m) = 20*logsig(wNN(:,mm)'*testCl(:,m)+bNN(mm))-10;
   end
   testProj = [testProj o(:,m)];
   diff = [];
```

```
   for mm = 1:num_class
      dist = o(:,m) - r(:,mm);
      diff = [diff dist'*dist];
   end
   [y index] = min(diff,[],2);
   classNumber = ceil(m/testData);
   type1(classNumber,index) = type1(classNumber,index) + 1;

   if (userReq == 'Y')
      disp([num2str(diff),'          ',num2str(index),'     ',num2str(y)])
      if (mod(m,testData)==0)
         disp('*****')
      end
   end
end
disp(' ')

% DISPLAY test data class typing
for m = 1:num_class+1
   disp(['type',num2str(m),': ',num2str(type1(m,:)),'       ',num2str(testData)])
end
disp(' ')

load type.dat
type = type + type1;
save type.dat type -ascii -tabs

%*****************************************************************************************
% PLOT class type identifier and test data projections
% NOTE:  1.  can only plot first three features
%        2.  testProj also includes projection of non-
%            class data
if (gloUsrReq == 'N')
   userReq = input('Plot projections (Y/N):  ','s');
   if (userReq == 'Y')
      fig = fig+1;
      plotProjection(testProj(1:3,:),r(1:3,:),testData,method,fig)
   end
   disp(' ')
end

% DISPLAY class type identifier and test data projections
% NOTE:   testProj also includes projection of non-class data
if (gloUsrReq == 'N')
   userReq = input('Display projection data (Y/N):  ','s');
   if (userReq == 'Y')
      dispProjection(testProj,r,testData,method)
   end
   disp(' ')
end

return
```

b. logsig.m

```
function a = logsig(n,b)

% where to put: c:\matlab\work\test
%LOGSIG Log sigmoid transfer function.
%
%       LOGSIG(N)
%         N - SxQ Matrix of net input (column) vectors.
%       Returns the values of N squashed between 0 and 1.
%
%       EXAMPLE: n = -10:0.1:10;
%                a = logsig(n);
```

```
%               plot(n,a)
%
%       LOGSIG(Z,B) ...Used when Batching.
%         Z - SxQ Matrix of weighted input (column) vectors.
%         B - Sx1 Bias (column) vector.
%       Returns the squashed net input values found by adding
%         B to each column of Z.
%
%       LOGSIG('delta') returns name of delta function.
%       LOGSIG('init') returns name of initialization function.
%       LOGSIG('name') returns full name of this transfer function.
%       LOGSIG('output') returns output range of this function.
%
%       See also NNTRANS, BACKPROP, NWTAN, LOGSIG.

% Mark Beale, 1-31-92
% Revised 12-15-93, MB
% Copyright (c) 1992-94 by The MathWorks, Inc.
% $Revision: 1.1 $  $Date: 1994/01/11 16:25:39 $

if nargin < 1, error('Note enough arguments.'); end

if isstr(n)
  if strcmp(lower(n),'delta')
    a = 'deltalog';
  elseif strcmp(lower(n),'init')
    a = 'nwlog';
  elseif strcmp(lower(n),'name')
    a = 'Log Sigmoid';
  elseif strcmp(lower(n),'output')
    a = [0 1];
  else
    error('Unrecognized property.')
  end
else
  if nargin==2
    [nr,nc] = size(n);
    n = n + b*ones(1,nc);
  end
  a = 1 ./ (1+exp(-n));
end
```

c. *sigderiv.m*

```
function d=sigderiv(n)

%**************************************************************
% This function calculated the derivative of logsig function
% where to put: c:\matlab\work\test
%**************************************************************

d=exp(-n)./((1+exp(-n)).^2);
i = find(~finite(d));
d(i) = 0;
```

4. Standard Mean Separator

a. *trms_sp.m*

```
function [w1,b1] = trms_sp(w1,b1,p,tp,method,fig)

%*****************************************************************************************
% Function
%    TRAINS the mean separator neural network with performance parameter defined as
%      MD = -[E{20*logsig(w'*x+b)-10} - E{20*logsig(w'*y+b)-10}]^2
%    to determine weight and bias for optimal projection
```

```
%
% Use:  [w1,b1] = trms_sp(w1,b1,p,tp,fig)
%
% Input    w1:          initial weight vector (3x1)
%          b1:          initial bias (1x1)
%          p:           matrix of training data for two classes
%          tp:          training parameters (see below)
%          method:      mean separator variation number
%                       1 - standard
%                       2 - preconditioned input (Mod 1)
%                       5 - normalized projection (Mod 2)
                        8 - with VMR termination (Mod 3)
%          fig:         figure number
%
% Returns  w1:          optimized weight vector
%          b1:          optimized bias
%
% 26 February 2000
% Miguel G. San Pedro
%*************************************************************************************
% MEAN SEPARATOR training function
% GENERAL EQUATION
%    MD(w,b) = -[mean(20*logsig{w'*x+b}-10) - mean(20*logsig{w'*y+b}-10)]^2
%            = -[20*mean(logsig{w'*x+b})-10 - 20*mean(logsig{w'*y+b}) + 10]^2
%            = -400[mean(logsig{w'*x+b})- mean(logsig{w'*y+b})]^2
%            = -400[mean(logsig{w'*x+b} - logsig{w'*y+b})]^2
%
% DETERMINE gradient by
% dMD/dw = c*d1
% with c  = -800[mean(logsig{w'*x+b} - logsig{w'*y+b})]
%       d1 = mean(der_logsig{w'*x+b}*x-der_logsig{w'*y+b}*y,2)
%
% dMD/db = c*d2
% with d2 = mean(der_logsig{w'*x+b}-der_logsig{w'*y+b})
%
% Training parameters(tp)
%         tp(1):  epochs between updating display
%         tp(2):  maximum number of epochs to train
%         tp(3):  initial lerning rate
%         tp(4):  learning rate increase
%         tp(5):  learning rate decrease
%         tp(6):  momentum constant
%         tp(7):  maximum error ratio
%
%*************************************************************************************

global gloUsrReq
global gloUsrPlot

% TRAINING PARAMETERS
df = tp(1);
me = tp(2);
lr = tp(3);
im = tp(4);
dm = tp(5);
mc = tp(6);
er = tp(7);

dw1 = 0;
db1 = 0;
MC = 0;
[pRow,pCol] = size(p);

nx = zeros(pRow/2,pCol);
ny = nx;
nx(1:pRow/2,:) = p(1:pRow/2,:);
ny(1:pRow/2,:) = p(1+pRow/2:pRow,:);
```

```
logsig_x = logsig(w1'*nx+b1);
logsig_y = logsig(w1'*ny+b1);

a = -400*(mean(logsig_x - logsig_y,2))^2;

% CHECK how weights and bias are changing
%load ..\checkWB.dat

% TRAINING
if (gloUsrReq == 'N')
   userReq = input('Display PROJ_INDEX update message (Y/N):        ','s');
else
   userReq = 'N';
end
if (userReq == 'Y')
   message = sprintf('TRAINMSNN: %%g/%g epochs, PROJ_INDEX = %%g.\n',me);
   fprintf(message,0,a)
   disp(['lr = ',num2str(lr)])
end

ctr_repeat = 0;
go_on = 1;
ii = 1;
a_save = 0;
plot_a_save = 0;
plot_lr_save = 0;
w1_save = rand(pRow/2,1);
b1_save = rand(1);
while(go_on==1)
   % LEARNING PHASE
   [dw1,db1] = lrms_sp(w1,b1,p,dw1,db1,lr,MC);

   % stepsize (alpha in steepest descent algorithm) incorporated as last step in lrms_sp
   new_w1 = w1-dw1;
   new_b1 = b1-db1;
   new_a = -400*(mean(logsig(new_w1'*nx+new_b1) - logsig(new_w1'*ny+new_b1),2))^2;
   MC = mc;

   % PRESENTATION PHASE
   if (new_a > a/er)
      lr = lr*dm;
      MC = 0;
   else
      if (new_a < a)
         lr = lr*im;
      end
      w1 = new_w1;
      b1 = new_b1;
      a = new_a;
   end
%   checkWB =[checkWB; [a w1' b1]];

   % TRAINING RECORD
   % PLOTTING
   plot_a(ii) = a;
   plot_lr(ii) = lr;

   % DISPLAY performance parameter
   if (userReq == 'Y')
      if (rem(ii,df) == 0)
         fprintf(message,ii,a)
         disp(['lr = ',num2str(lr)])
      end
   end

% if lr falls below minimum allowable (no learning being accomplished), break out of loop
```

```
% if final MD > -360, reset loop counter, choose new initial weights and bias and repeat
% loop
   if ((lr < 1e-4)|(ii == me))
      if (abs(a_save) < abs(a))
         a_save = a;
         w1_save = w1;
         b1_save = b1;
         plot_a_save = plot_a;
         plot_lr_save = plot_lr;
      end
      if ((a_save > -360)&(ctr_repeat <= 10))
         ii = 0;
         plot_a = [];
         plot_lr = [];
         w1 = randn(pRow/2,1);
         b1 = randn(1,1);

         a = -400*(mean(logsig(w1'*nx+b1) - logsig(w1'*ny+b1),2))^2;

         dw1 = 0;
         db1 = 0;
         MC = 0;
         lr = tp(3);
         ctr_repeat = ctr_repeat+1;
%         checkWB = [checkWB; 0001 zeros(size(w1')) NaN];
         if (userReq == 'Y')
            disp('*** INSUFFICIENT PROJECTION INDEX ***')
            disp(' ')
         end
      else
         go_on = 0;
      end
   end
   ii = ii+1;
end

disp(['num epochs =',num2str(ii-1)])
disp(['lr = ',num2str(lr)])
disp(['MD = ',num2str(a_save)])

w1 = w1_save;
b1 = b1_save;
disp(' ')

if (gloUsrPlot == 'Y')
   figure(fig)
   orient tall
   subplot(211)
   plot(plot_a_save)
   xlabel('time')
   ylabel('MD')
   title(['MD vs time (Method',num2str(method),')'])
   grid on

   subplot(212)
   plot(plot_lr_save)
   xlabel('time')
   ylabel('lr')
   title(['learning rate vs time (Method',num2str(method),')'])
   grid on
end

%checkWB = [checkWB; 0001 ones(size(w1')) NaN];
%save ..\checkWB.dat checkWB -ascii -tabs

return
```

b. lrms_sp.m

```
function [dw,db] = lrms_sp(w,b,p,dw1,db1,lr,mc)

%****************************************************************************************
% Function
%    Learning rate function for the mean separator neural network with performance
%    parameter defined as
%       MD = -[E{20*logsig(w'*x+b)-10} - E{20*logsig(w'*y+b)-10}]^2
%    to determine change in weight and bias for optimal projection
%
% Use:   [dw,db] = lrms_sp(w,b,p,dw1,db1,lr,mc)
%
% Input     w:          weight vector (3x1)
%           b:          bias (1x1)
%           p:          matrix of training data for two classes
%           dw1:        current change in weight
%           db1:        current change in bias
%           lr:         learning rate
%           mc:         momentum constant
%
% Returns   dw:         weight vector change (3x1)
%           db:         bias change (1x1)
%
% 16 January 2000
% Miguel G. San Pedro
%****************************************************************************************

[pRow,pCol] = size(p);
nx = zeros(pRow/2,pCol);
ny = nx;
nx(1:pRow/2,:) = p(1:pRow/2,:);
ny(1:pRow/2,:) = p(pRow/2+1:pRow,:);

logsig_x = logsig(w'*nx+b);
logsig_y = logsig(w'*ny+b);
der_logsig_x = sigderiv(w'*nx+b);
der_logsig_y = sigderiv(w'*ny+b);

d11 = [];
d11 = der_logsig_x([ones(1,pRow/2)],:);
d12 = [];
d12 = der_logsig_y([ones(1,pRow/2)],:);
d1 = mean(d11.*nx - d12.*ny,2);

c = -800*(mean(logsig_x,2) - mean(logsig_y,2));
dw = c*d1;
db = c*mean(der_logsig_x - der_logsig_y,2);

% APPLY adaptive lr and stepsize
dw = mc*dw1 + (1-mc)*lr*dw;
db = mc*db1 + (1-mc)*lr*db;

return
```

c. meansep_sp1

```
function a = meansep_sp1(p1,p2,w,b)

%****************************************************************************************
% Function
%    CALCULATES the mean separator neural network with performance parameter defined as
%       MD(w,b) = -[mean(20*logsig(w'*x+b)-10) - mean{20*logsig(w'*y+b)-10}]^2
%
% Use:   a = meansep_sp1(p1,p2,w,b)
%
```

```
% Input     p1:          row feature vector for first class
%           p2:          row feature vector for second class
%           w:           weight vector
%           b:           bias
%
% Returns   a:           mean separator performance parameter value
%
% 5 January 2000
% Miguel G. San Pedro
%*************************************************************************************

if nargin < 3, error('Not enough arguments.'); end

alpha = logsig(w'*p1 + b);
beta = logsig(w'*p2 + b);
a = -400*(mean(alpha - beta,2))^2;

return
```

5. Preconditioned Input Data (MSNN Mod 1): *simmsnn_C.m*

```
function simmsnn_C(classData_norm,num_features,w,b,tp,fig)

%*************************************************************************************
% Function
%     SIMULATES the mean separator neural network with performance parameter defined as
%        MD = -[E{20*logsig(w'*[(x-mean(x))/sd(x)+mean(x)]+b)-10}
%                - E{20*logsig(w'*[(y-mean(y))/sd(y)+mean(y)]+b)-10}]^2
%
% Use: simmsnn_C(classData_norm,num_features,w,b,tp,fig)
%
% Calls    trms_sp and lrms_sp since equations are same; only input vectors differ
%
% Input     classData_norm:    normalized training data
%           w:                 projection weight vector
%           b:                 projection bias
%           tp:                training parameters (see function trms_sp2)
%           fig:               figure number
%
% Returns   (none)
%
% 23 February 2000
% Miguel G. San Pedro
%*************************************************************************************
global gloUsrReq

method = 2;

[classRow,num_data] = size(classData_norm);
num_class = classRow/num_features;
num_prwise = sum(1:num_class-1);       % number of pairwise comparisons
ind = 0;                               % pairwise index
r = zeros(num_prwise,num_class);       % class type identifier

%*************************************************************************************
% COMPARE class k and class kk
for k = 1:num_class
   knum_feat = k*num_features;
   for kk = k+1:num_class
      kknum_feat = kk*num_features;
      ind = ind + 1;

      class1 = classData_norm(knum_feat-num_features+1:knum_feat,:);
      class2 = classData_norm(kknum_feat-num_features+1:kknum_feat,:);
      p1 = [class1;class2];
```

```
      disp(['Class ',num2str(k),' vs Class ',num2str(kk)])
      fig = fig+1;
      [wNN(:,ind),bNN(ind)] = trms_sp(w,b,p1,tp,method,fig);

      % DETERMINE class type identifier for this pairwise comparison
      for mm = 1:num_class
         mmnum_feat = mm*num_features;
         classA = classData_norm(mmnum_feat-num_features+1:mmnum_feat,:);
         r(ind,mm) = 20*mean(logsig(wNN(:,ind)'*classA + bNN(ind)))-10;

        % DETERMINE projection data for neuron maps
         plotr = [plotr 20*logsig(wNN(:,ind)'*classA + bNN(ind))-10];

      end

   % PLOT neuron maps
   figure
   plot(plotr)
   xlabel('Test Point')
   ylabel(['Neuron Map [',num2str(k),',',num2str(kk),']'])

   end
end

% DISPLAY weights/bias and class type identifiers
if (gloUsrReq == 'N')
   userReq = input('Display projection weights and biases (Y/N):    ','s');
   if (userReq == 'Y')
      dispWeightBias(wNN,bNN)
   end
   disp(' ')
end

%*******************************************************************************************
% CLASSIFY test points
load ..\test\testClass_norm.dat
[testRow,testData] = size(testClass_norm);

numTestData = testData/(num_class+1);
if (testRow ~= num_features*num_class)
   disp('*** DATA ERROR')
end

% PROJECT/TYPE testClass data
% 'diff' matrices store distances from class type identifiers (r's) to data projections
% (o's) determine best fit (i.e. trial data typing) by deteriming minimum value of each
% row
% 2nd dimension of r gives number of classes, testData gives number of test data points
% taking column number of each testProj point and performing ceil(colNum/testData) gives
% class number

type1 = zeros(num_class+1,num_class);

if (gloUsrReq == 'N')
   userReq = input('Display typing distance data (Y/N):     ','s');
else
   userReq = 'N';
end

diffMat = [];
for k = 1:num_class
   knum_feat = k*num_features;
   xk = [knum_feat - num_features + 1:knum_feat];
   diffRow = [];
   for kk = 1:testData
      for mm = 1:num_prwise
         o(mm,kk) = 20*logsig(wNN(:,mm)'*testClass_norm(xk,kk))-10;
```

```
      end
      dist = o(:,kk) - r(:,k);
      diffRow = [diffRow dist'*dist];
   end
   diffMat = [diffMat;diffRow];
end
[y index] = min(diffMat,[],1);

for k = 1:num_class+1
   for kk = 1:numTestData
      xx = (k-1)*numTestData+kk;
      type1(k,index(xx)) = type1(k,index(xx))+1;
   end
end
disp(' ')

% DISPLAY test data class typing
for m = 1:num_class+1
   disp(['type',num2str(m),': ',num2str(type1(m,:))])
end

load type.dat
type = type + type1;
save type.dat type -ascii -tabs

%*******************************************************************************************
% PLOT class type identifier and test data projections - option not permitted

%*******************************************************************************************
% PLOT class type identifier and test data projections - option not permitted

return
```

6. Normalized Projection Space (MSNN Mod 2)

a. *trms_sp5.m*

```
function [w1,b1] = trms_sp5(w1,b1,p,tp,method,fig)

%*******************************************************************************************
% Function
%    TRAINS the mean separator neural network with performance parameter defined as
%       MD = -[E{alpha - beta}]^2*[E{(alpha - E{alpha})^2}
%                                         + E{(beta - E{beta})^2} + delta]^-1
%    with alpha = logsig(w'*x+b), beta = logsig(w'*y+b), and delta precludes division by
%    zero, to determine weight and bias for optimal projection
%    NORMALIZES basic performance parameter (standard MSNN) by sum of projection
%    variances
%
% Use:  [w1,b1] = trms_sp5(w1,b1,p,tp,method,fig)
%
% Input    w1:          initial weight vector (3x1)
%          b1:          initial bias (1x1)
%          p:           matrix of training data for two classes
%          tp:          training parameters (see below)
%          method:      mean separator variation number
%                       1 - standard
%                       2 - preconditioned input (Mod 1)
%                       5 - normalized projection (Mod 2)
%                       8 - with VMR termination (Mod 3)
%          fig:         figure number
%
% Returns  w1:          optimized weight vector
%          b1:          optimized bias
%
% 26 February 2000
```

```
% Miguel G. San Pedro
%***************************************************************************************
% MEAN SEPARATOR training function
% GENERAL EQUATION
%    MD(w,b) = -[E{20*logsig(w'*x+b)-10} - E{20*logsig(w'*y+b)-10}]^2
%                 *[var(20*logsig(w'x+b)-10) +.var(20*logsig(w'y+b)-10) + delta]^-1
%            = -[E{20*logsig(w'*x+b)-10} - E{20*logsig(w'*y+b)-10}]^2
%              *[E{(20*logsig(w'*x+b)-10 - E{20*logsig(w'*x+b)-10})^2
%                + E{(20*logsig(w'*y+b)-10 - E{20*logsig(w'*y+b)-10})^2 + delta]^-1
%            = -[20*E{logsig(w'*x+b)}-10 - 20*E{logsig(w'*y+b)+10}]^2
%              *[E{(20*logsig(w'*x+b)-10 - 20*E{logsig(w'*x+b)}+10)^2
%                + E{(20*logsig(w'*y+b)-10 - 20*E{logsig(w'*y+b)}+10)^2 + delta]^-1
%            = -[E{logsig(w'*x+b)} - E{logsig(w'*y+b)}]^2
%              *[E{(logsig(w'*x+b) - E{logsig(w'*x+b)})^2}
%               + E{(logsig(w'*y+b) - E{logsig(w'*y+b)})^2} + delta]^-1
%     let alpha = logsig(w'*x+b), beta = logsig(w'*y+b)
%            = -[E{alpha} - E{beta}]^2*[E{(alpha - E{alpha})^2} + [E{(beta - E{beta})^2}
%                                + delta]^1
%            = -[E{alpha - beta}]^2*[E{alpha^2 + beta^2}
%                              - E^2{alpha} - E^2{beta} + delta]^-1
%     or, alpha = -[E{alpha - beta}]^2/[var(alpha) + var(beta) + delta]
%     note:  if den is infinitesimally small, delta = 1e-10
%
% DETERMINE gradient by
%    K       = E{alpha - beta}/(E{alpha^2 + beta^2} - E^2{alpha} - E^2{beta} + delta)
%    dMD/dw  = 2K[K*(E{alpha*dalpha/dw + beta*dbeta/dw}
%                      - E{alpha}E{dalpha/dw} - E{beta}E{dbeta/dw})
%                      - E{dalpha/dw - dbeta/dw}]
%    dMD/db  = 2K[K*(E{alpha*dalpha/db + beta*dbeta/db}
%                      - E{alpha}E{dalpha/db} - E{beta}E{dbeta/db})
%                      - E{dalpha/db - dbeta/db}]
%
% Training parameters(tp)
%        tp(1):  epochs between updating display
%        tp(2):  maximum number of epochs to train
%        tp(3):  initial lerning rate
%        tp(4):  learning rate increase
%        tp(5):  learning rate decrease
%        tp(6):  momentum constant
%        tp(7):  maximum error ratio
%
%***************************************************************************************
global gloUsrReq
global gloUsrPlot

format short e
delta = 1e-10;

% TRAINING PARAMETERS
df = tp(1);
me = tp(2);
lr = tp(3);
im = tp(4);
dm = tp(5);
mc = tp(6);
er = tp(7);

dw1 = 0;
db1 = 0;
MC = 0;
[pRow,pCol] = size(p);

nx = zeros(pRow/2,pCol);
ny = nx;
nx(1:pRow/2,:) = p(1:pRow/2,:);
ny(1:pRow/2,:) = p(1+pRow/2:pRow,:);
```

```
alpha = logsig(w1'*nx+b1);
beta = logsig(w1'*ny+b1);

E_alpha = mean(alpha,2);
E_beta = mean(beta,2);
var_alpha = var(alpha,1);
var_beta = var(beta,1);

num = (E_alpha - E_beta)^2;
den = var_alpha + var_beta;
if (den < 1e-10)
   den = delta;
end
a = -num/den;

% CHECK how mean and variance are updating
%checkMD = [];
%checkMD = [checkMD; [num den]];

% CHECK how weights and bias are changing
%load ..\checkWB.dat

% TRAINING
if (gloUsrReq == 'N')
   userReq = input('Display PROJ_INDEX update message (Y/N):        ','s');
else
   userReq = 'N';
end
if (userReq == 'Y')
   message = sprintf('TRAINMSNN: %%g/%g epochs, PROJ_INDEX = %%g.\n',me);
   fprintf(message,0,a)
   disp(['lr = ',num2str(lr)])
end

ctr_repeat = 0;
go_on = 1;
ii = 1;
a_save = 0;
plot_a_save = [];
plot_lr_save = [];
w1_save = rand(pRow/2,1);
b1_save = rand(1);
GOODcheck = 0;

while(go_on==1)
   % LEARNING PHASE
   [dw1,db1] = lrms_sp5(w1,b1,p,dw1,db1,lr,MC);

   % stepsize (alpha in steepest descent algorithm) incorporated as
   % last step in lrms_sp5
   new_w1 = w1-dw1;
   new_b1 = b1-db1;

   new_alpha = logsig(new_w1'*nx+new_b1);
   new_beta = logsig(new_w1'*ny+new_b1);

   E_new_alpha = mean(new_alpha,2);
   E_new_beta = mean(new_beta,2);
   var_new_alpha = var(new_alpha,1);
   var_new_beta = var(new_beta,1);

   new_num = (E_new_alpha - E_new_beta)^2;
   new_den = var_new_alpha + var_new_beta;
   if (new_den < 1e-10)
      new_den = delta;
   end
   new_a = -new_num/new_den;
```

```
   MC = mc;

   % PRESENTATION PHASE
   if (new_a > a/er)
      lr = lr*dm;
      MC = 0;
   else
      if (new_a < a)
         lr = lr*im;
      end
      w1 = new_w1;
      b1 = new_b1;
      a = new_a;
      num = new_num;
      den = new_den;
   end
%   checkWB =[checkWB; [a w1' b1]];
%   checkMD = [checkMD; [num den]];

   % TRAINING RECORD
   % PLOTTING
   plot_a(ii) = a;
   plot_lr(ii) = lr;

   % DISPLAY performance parameter
   if (userReq == 'Y')
      if (rem(ii,df) == 0)
         fprintf(message,ii,a)
         disp(['lr = ',num2str(lr)])
      end
   end

% CHECK improvement in performance parameter
   if (abs(a_save) < abs(a))
      a_save = a;
      w1_save = w1;
      b1_save = b1;
      plot_a_save = plot_a;
      plot_lr_save = plot_lr;
      lr = lr/0.9;                          % prevents stalling training trajectory

      % CALCULATE termination parameter
      % Termination parameter:  considered with ratio of difference in Q(+-0.005) pts
      %                        and difference of means
      % Assume Gaussian distribution
      % 1.65 gives 5.0% in tails
      % 1.95 gives 2.5% in tails
      % 2.52 gives 0.5% in tails
      GOOD_alpha = logsig(w1_save'*nx+b1_save);
      GOOD_beta = logsig(w1_save'*ny+b1_save);

      E_GOOD_alpha = mean(GOOD_alpha,2);
      E_GOOD_beta = mean(GOOD_beta,2);
      var_GOOD_alpha = var(GOOD_alpha,1);
      var_GOOD_beta = var(GOOD_beta,1);

      GOODcheck = 1 - 2.52*(sqrt(var_GOOD_alpha) + sqrt(var_GOOD_beta))...
                              /abs(E_GOOD_alpha - E_GOOD_beta);
   end

   if ((lr < 1e-4)|(ii == me)|(GOODcheck > 0.90))
      go_on = 0;
   end
   ii = ii+1;                               % INCREMENT epoch counter
end
disp(['num epochs =',num2str(ii-1)])
```

```
disp(['lr = ',num2str(lr)])
disp(['MD = ',num2str(a_save)])
disp(['VMR = ',num2str(GOODcheck)])

w1 = w1_save;
b1 = b1_save;
disp (' ')

if (gloUsrPlot == 'Y')
   figure(fig)
   orient tall
   subplot(211)
   plot(plot_a_save)
   xlabel('time')
   ylabel('MD')
   title(['MD vs time (Method',num2str(method),')'])
   grid on

   subplot(212)
   plot(plot_lr_save)
   xlabel('time')
   ylabel('lr')
   title(['learning rate vs time (Method',num2str(method),')'])
   grid on
end

%checkWB = [checkWB; 0005 ones(size(w1')) NaN];
%save ..\checkWB.dat checkWB -ascii -tabs

%save checkMD.dat  checkMD -ascii -tabs

return
```

b. lrms_sp5.m

```
function [dw,db] = lrms_sp5(w,b,p,dw1,db1,lr,mc)

%***************************************************************************************
% Function
%    Learning rate function for the mean separator neural network with performance
%    parameter defined as
%       MD = -[E{alpha - beta}]^2*[E{(alpha - E{alpha})^2} + E{(beta - E{beta})^2}
%                                     + delta]^-1
%    with alpha = logsig(w'*x+b), beta = logsig(w'*y+b), and delta precludes division by
%    zero
%     note:  if den is infinitesimally small, delta = 1e-10
%    Determines change in weight and bias for optimal projection
%
% Use:   [dw,db] = lrms_sp5(w,b,p,dw1,db1,lr,mc)
%
% Input      w:           weight vector (3x1)
%            b:           bias (1x1)
%            p:           matrix of training data for two classes
%            dw1:         current change in weight
%            db1:         current change in bias
%            lr:          learning rate
%            mc:          momentum constant
%
% Returns    dw:          weight vector change (3x1)
%            db:          bias change (1x1)
%
% 16 January 2000
% Miguel G. San Pedro
%***************************************************************************************
delta = 1e-10;
```

```
[pRow,pCol] = size(p);
nx = zeros(pRow/2,pCol);
ny = nx;
nx(1:pRow/2,:) = p(1:pRow/2,:);
ny(1:pRow/2,:) = p(1+pRow/2:pRow,:);

alpha = logsig(w'*nx+b);
E_alpha = mean(alpha,2);
beta = logsig(w'*ny+b);
E_beta = mean(beta,2);

dalpha_db = sigderiv(w'*nx+b);
E_dalpha_db = mean(dalpha_db,2);
dbeta_db = sigderiv(w'*ny+b);
E_dbeta_db = mean(dbeta_db,2);

dx = [];
dx = dalpha_db([ones(1,pRow/2)],:);
dy = [];
dy = dbeta_db([ones(1,pRow/2)],:);

dalpha_dw = dx.*nx;
E_dalpha_dw = mean(dalpha_dw,2);
dbeta_dw = dy.*ny;
E_dbeta_dw = mean(dbeta_dw,2);

alpha_mat = [];
alpha_mat = alpha([ones(1,pRow/2)],:);
beta_mat = [];
beta_mat = beta([ones(1,pRow/2)],:);
den = var(alpha,1) + var(beta,1);
if (den < 1e-10)
   den = delta;
end
K = mean(alpha-beta,2)/den;
dw = 2*K*(K*(mean(alpha_mat.*dalpha_dw+beta_mat.*dbeta_dw,2)...
        - E_alpha*E_dalpha_dw - E_beta*E_dbeta_dw) - E_dalpha_dw + E_dbeta_dw);
db = 2*K*(K*(mean(alpha.*dalpha_db+beta.*dbeta_db,2)...
        - E_alpha*E_dalpha_db - E_beta*E_dbeta_db) - E_dalpha_db + E_dbeta_db);

% APPLY adaptive lr and stepsize
dw = mc*dw + (1-mc)*lr*dw;
db = mc*db + (1-mc)*lr*db;

return
```

c. meansep_sp5.m

```
function a = meansep_sp5(p1,p2,w,b)

%*****************************************************************************************
% Function
%    CALCULATES the mean separator neural network with performance parameter defined as
%      MD = -[E{alpha - beta}]^2*[E{(alpha - E{alpha})^2}
%                         + E{(beta - E{beta})^2} + delta]^-1
%    with alpha = logsig(w'*x+b), beta = logsig(w'*y+b), and delta precludes division by
%    zero
%     note:  if den is infinitesimally small, delta = 1e-10
%    NORMALIZES basic performance parameter (Method1) by sum of projection variances
%
% Use:  a = meansep_sp5(p1,p2,w,b)
%
% Input     p1:         matrix of features for first class
%           p2:         matrix of features for second class
%           w:          weight vector
%           b:          bias
```

```
%
% Returns  a:          mean separator performance parameter value
%
% 5 January 2000
% Miguel G. San Pedro
%****************************************************************************************

if nargin < 3, error('Not enough arguments.'); end

delta = 1e-10;

alpha = logsig(w'*p1 + b);
beta = logsig(w'*p2 + b);
num = (mean(alpha - beta,2))^2;
den = var(alpha) + var(beta);

if (den < 1e-10)
   den = delta;
end

a = -num/den;

return
```

7. Standard MSNN with VMR Termination (MSNN Mod 3)

a. trms_sp8.m

```
function [w1,b1] = trms_sp5(w1,b1,p,tp,method,fig)

%****************************************************************************************
% Function
%    TRAINS the mean separator neural network with performance parameter defined as
%       MD = -[E{20*logsig(w'*x+b)-10} - E{20*logsig(w'*y+b)-10}]^2
%    to determine weight and bias for optimal projection
%
% Use:  [w1,b1] = trms_sp8(w1,b1,p,tp,method,fig)
%
% Input     w1:          initial weight vector (3x1)
%           b1:          initial bias (1x1)
%           p:           matrix of training data for two classes
%           tp:          training parameters (see below)
%           method:      mean separator variation number
%                        1 - standard
%                        2 - preconditioned input (Mod 1)
%                        5 - normalized projection (Mod 2)
%                        8 - with VMR termination (Mod 3)
%           fig:         figure number
%
% Returns   w1:          optimized weight vector
%           b1:          optimized bias
%
% 26 February 2000
% Miguel G. San Pedro
%****************************************************************************************
% MEAN SEPARATOR training function
% GENERAL EQUATION
%    MD(w,b) = -[mean(20*logsig{w'*x+b}-10) - mean(20*logsig(w'*y+b)-10}]^2
%            = -[20*mean(logsig{w'*x+b})-10 - 20*mean(logsig{w'*y+b}) + 10]^2
%            = -400[mean(logsig{w'*x+b})- mean(logsig{w'*y+b})]^2
%            = -400[mean(logsig{w'*x+b} - logsig{w'*y+b})]^2
%
% DETERMINE gradient by
% dMD/dw = c*d1
% with c  = -800[mean(logsig{w'*x+b} - logsig{w'*y+b})]
%      d1 = mean(der_logsig{w'*x+b}*x-der_logsig{w'*y+b}*y,2)
```

```
%
% dMD/db = c*d2
% with d2 = mean(der_logsig{w'*x+b}-der_logsig{w'*y+b})
%
% Training parameters(tp)
%         tp(1):  epochs between updating display
%         tp(2):  maximum number of epochs to train
%         tp(3):  initial lerning rate
%         tp(4):  learning rate increase
%         tp(5):  learning rate decrease
%         tp(6):  momentum constant
%         tp(7):  maximum error ratio
%
%*******************************************************************************************
global gloUsrReq
global gloUsrPlot

format short e
delta = 1e-10;

% TRAINING PARAMETERS
df = tp(1);
me = tp(2);
lr = tp(3);
im = tp(4);
dm = tp(5);
mc = tp(6);
er = tp(7);

dw1 = 0;
db1 = 0;
MC = 0;
[pRow,pCol] = size(p);

nx = zeros(pRow/2,pCol);
ny = nx;
nx(1:pRow/2,:) = p(1:pRow/2,:);
ny(1:pRow/2,:) = p(1+pRow/2:pRow,:);

alpha = logsig(w1'*nx+b1);
beta = logsig(w1'*ny+b1);

E_alpha = mean(alpha,2);
E_beta = mean(beta,2);

a = -(E_alpha - E_beta)^2;

% CHECK how weights and bias are changing
%load ..\checkWB.dat

% TRAINING
if (gloUsrReq == 'N')
   userReq = input('Display PROJ_INDEX update message (Y/N):        ','s');
else
   userReq = 'N';
end
if (userReq == 'Y')
   message = sprintf('TRAINMSNN: %%g/%g epochs, PROJ_INDEX = %%g.\n',me);
   fprintf(message,0,a)
   disp(['lr = ',num2str(lr)])
end

ctr_repeat = 0;
go_on = 1;
ii = 1;
a_save = 0;
plot_a_save = [];
```

```
plot_lr_save = [];
w1_save = rand(pRow/2,1);
b1_save = rand(1);
GOODcheck = 0;

while(go_on==1)
   % LEARNING PHASE
   [dw1,db1] = lrms_sp8(w1,b1,p,dw1,db1,lr,MC);

   % stepsize (alpha in steepest descent algorithm) incorporated as
   % last step in lrms_sp8
   new_w1 = w1-dw1;
   new_b1 = b1-db1;

   new_alpha = logsig(new_w1'*nx+new_b1);
   new_beta = logsig(new_w1'*ny+new_b1);

   E_new_alpha = mean(new_alpha,2);
   E_new_beta = mean(new_beta,2);

   new_num = (E_new_alpha - E_new_beta)^2;
   new_a = -new_num;

   MC = mc;

   % PRESENTATION PHASE
   if (new_a > a/er)
      lr = lr*dm;
      MC = 0;
   else
      if (new_a < a)
         lr = lr*im;
      end
      w1 = new_w1;
      b1 = new_b1;
      a = new_a;
end
%   checkWB =[checkWB; [a w1' b1]];
%   checkMD = [checkMD; [num den]];

   % TRAINING RECORD
   % PLOTTING
   plot_a(ii) = a;
   plot_lr(ii) = lr;

   % DISPLAY performance parameter
   if (userReq == 'Y')
      if (rem(ii,df) == 0)
         fprintf(message,ii,a)
         disp(['lr = ',num2str(lr)])
      end
   end

% CHECK improvement in performance parameter
   if (abs(a_save) < abs(a))
      a_save = a;
      w1_save = w1;
      b1_save = b1;
      plot_a_save = plot_a;
      plot_lr_save = plot_lr;
      lr = lr/0.9;                          % prevents stalling training trajectory

      % CALCULATE termination parameter
      % Termination paramter:  considered with ratio of difference in Q(+-0.005) pts
      %                        and difference of means
      % Assume Gaussian distribution
      % 1.65 gives 5.0% in tails
```

```
      % 1.95 gives 2.5% in tails
      % 2.52 gives 0.5% in tails
      GOOD_alpha = logsig(w1_save'*nx+b1_save);
      GOOD_beta = logsig(w1_save'*ny+b1_save);

      E_GOOD_alpha = mean(GOOD_alpha,2);
      E_GOOD_beta = mean(GOOD_beta,2);
      var_GOOD_alpha = var(GOOD_alpha,1);
      var_GOOD_beta = var(GOOD_beta,1);

      GOODcheck = 1 - 2.52*(sqrt(var_GOOD_alpha) + sqrt(var_GOOD_beta))...
                              /abs(E_GOOD_alpha - E_GOOD_beta);
   end

   if ((lr < 1e-4)|(ii == me)|(GOODcheck > 0.90))
      go_on = 0;
   end
   ii = ii+1;                                  % INCREMENT epoch counter
end
disp(['num epochs =',num2str(ii-1)])
disp(['lr = ',num2str(lr)])
disp(['MD = ',num2str(a_save)])
disp(['VMR = ',num2str(GOODcheck)])

w1 = w1_save;
b1 = b1_save;
disp (' ')

if (gloUsrPlot == 'Y')
   figure(fig)
   orient tall
   subplot(211)
   plot(plot_a_save)
   xlabel('time')
   ylabel('MD')
   title(['MD vs time (Method',num2str(method),')'])
   grid on

   subplot(212)
   plot(plot_lr_save)
   xlabel('time')
   ylabel('lr')
   title(['learning rate vs time (Method',num2str(method),')'])
   grid on
end

%checkWB = [checkWB; 0005 ones(size(w1')) NaN];
%save ..\checkWB.dat checkWB -ascii -tabs

%save checkMD.dat  checkMD -ascii -tabs

return
```

b. lrms_sp8.m

```
function [dw,db] = lrms_sp8(w,b,p,dw1,db1,lr,mc)

%*******************************************************************************************
% Function
%    Learning rate function for the mean separator neural network with performance
%    parameter defined as
%       MD = -[E{20*logsig(w'*x+b)-10} - E{20*logsig(w'*y+b)-10}]^2
%    to determine change in weight and bias for optimal projection
%
% Use:  [dw,db] = lrms_sp8(w,b,p,dw1,db1,lr,mc)
%
% Input    w:          weight vector (3x1)
```

```
%           b:          bias (1x1)
%           p:          matrix of training data for two classes
%           dw1:        current change in weight
%           db1:        current change in bias
%           lr:         learning rate
%           mc:         momentum constant
%
% Returns   dw:         weight vector change (3x1)
%           db:         bias change (1x1)
%
% 26 February 2000
% Miguel G. San Pedro
%**************************************************************************************

[pRow,pCol] = size(p);
nx = zeros(pRow/2,pCol);
ny = nx;
nx(1:pRow/2,:) = p(1:pRow/2,:);
ny(1:pRow/2,:) = p(pRow/2+1:pRow,:);

logsig_x = logsig(w'*nx+b);
logsig_y = logsig(w'*ny+b);
der_logsig_x = sigderiv(w'*nx+b);
der_logsig_y = sigderiv(w'*ny+b);

d11 = [];
d11 = der_logsig_x([ones(1,pRow/2)],:);
d12 = [];
d12 = der_logsig_y([ones(1,pRow/2)],:);
d1 = mean(d11.*nx - d12.*ny,2);

c = -800*(mean(logsig_x,2) - mean(logsig_y,2));
dw = c*d1;
db = c*mean(der_logsig_x - der_logsig_y,2);

% APPLY adaptive lr and stepsize
dw = mc*dw1 + (1-mc)*lr*dw;
db = mc*db1 + (1-mc)*lr*db;

return
```

LIST OF REFERENCES

Brunzell, H., "Extractions of Features for Classification of Impulse Radar Measurements," *SPIE Proceedings, Automatic Object Recognition VII*, **3069**, April 1997, pp. 321-330.

Brunzell, H. and Eriksson, J., "Feature Reduction for Classification of Multidimensional Data," draft, Dept of Electrical Engineering, The Ohio State University, 2 November 1999.

Dayhoff, Judith, *Neural Network Architecture: An Introduction*, New York, NY: Van Nostrand Reinhold, 1990.

Duzenli, Ozhan, *Classification of Underwater Signals Using Wavelet-Based Decompositions*, Master's Thesis, Naval Postgraduate School, Monterey, CA, June 1998.

Duzenli, Ozhan and Fargues, Monique P., "Wavelet-Based Feature Extraction Methods for Classification Applications," *Proceedings of 9th Signal Processing Workshop on Statistical Signals and Array Processing*, September 1998.

Fargues, Monique P. and Ozhan Duzenli, "Dimension Reduction Issues in Classification Applications," *Proceedings of 32nd Asilomar Conference on Signals, Systems, and Computers*, November 1998.

Fausett, Loren, *Fundamentals of Neural Networks: Architecture, Algorithms, and Application*, Upper Saddle River, NJ: Prentice-Hall, Inc., 1994.

Fukunaga, Keinosuke, *Introduction to Statistical Pattern Recognition*, San Diego, CA: Academic Press, 1990.

Ghani, Nasir and Lamontagne, Rene, "Neural Networks Applied to the Classification of Spectral Features for Automatic Modulation Recognition," *Proceedings of the 1993 Military Communication Conference*, **1**, 1993, pp. 111-115.

Hagan, Martin T., Demuth, Howard B., and Beale, Mark, *Neural Network Design*, Boston, MA: PWS Publishing, Co., 1996.

Haykin, Simon, *Neural Networks: A Comprehensive Foundation*, Englewood Cliffs, NJ: MacMillan Publishing Co., 1994.

Huo, Xiaoming and Donohuo, David, "A Simple and Robust Modulation Classification Method via Counting," *Proceedings of the 1998 IEEE International Conference on Acoustics, Speech and Signal Processing*, **6**, 1998, pp. 3289-3292.

Johnson, ADM Jay L., "A Maritime Strategy for the Naval Century," as quoted in **Campus News**, Naval Postgraduate School, Monterey, CA, **7**:7, February 17, 2000, p. 4.

Lallo, Pauli, "Signal Classification by Discrete Fourier Transform," *Proceedings of the 1999 Military Communications International Symposium*, MILCOM 1999.

MATLAB 5, ver 5.3.1, The MathWorks, Inc., Natick, MA, September 28, 1999.

Moravec, Hans, "When Will Computer Hardware Match the Human Brain?" *Journal of Transhumanism*, **1**, 1998, http://www.transhumanist.com/volume1/moravec.htm (active as of 8 April 1999).

Raudys, Sarunas, "Linear Classifiers in Perceptron Design," *Proceedings of the 13th International Conference on Pattern Recognition*, **3**, 1996, pp. 763-767.

Reichert, Juergen, "Automatic Classification of Communication Signals using Higher Order Statistics," *Proceedings of the 1992 IEEE International Conference on Acoustics, Speech, and Signal Processing*, **5**, 1992, pp. 221-224.

SAS Institute, Inc., "%MULTNORM macro: Mardia Tests of Multivariate Normality," 2000, http://ftp.sas.com/techsup/download/stat/multnorm.sas (active as of 27 February 2000).

Sills, J.A., "Maximum-Likelihood Modulation Classification for PSK/QAM," *Proceedings of the 1999 Military Communications International Symposium*, MILCOM 1999.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center 2
   8725 John J. Kingman Rd., Ste 0944
   Ft. Belvoir, VA 22060-6218

2. Dudley Knox Library 2
   Naval Postgraduate School
   411 Dyer Rd.
   Monterey, CA 93943-5101

3. Chairman, Code EC.. 1
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, CA 93943-5121

4. Engineering and Technology Curricular Office, Code 34 1
   Naval Postgraduate School
   Monterey, CA 93943-5107

5. Prof. Monique P. Fargues, Code EC/Fa 3
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, CA 93943-5121

6. Prof. Ralph D. Hippenstiel, Code EC/Hi 1
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, CA 93943-5121

7. Mr. William Glenney 1
   Deputy Director
   CNO Strategic Studies Group
   686 Cushing Road
   Newport, RI 02841-1207

8. LCDR Miguel G. San Pedro, U.S. Navy.. 2
   11916-8 Tivoli Park Row
   San Diego, CA 92128